EnFuzion 9.3 User Manual

Axceleon

EnFuzion 9.3 User Manual
by Axceleon

2nd Edition
Published January, 2009
Copyright © 2009 Axceleon, Inc.

Table of Contents

Preface ..... **i**

1. Overview of EnFuzion **1**

The Power of Many 1

Basic EnFuzion Concepts 1

Parametric Execution 2

Run 2

Job 3

Context 3

Submit Computers 3

Hardware and Software Requirements 3

Submit Configuration 3

Submit Processes 4

Job Submission and Retrieval of Results 4

Root Computers 4

Hardware and Software Requirements 4

Root Configuration 5

Root Processes 5

Root Monitoring and Control 5

Job Execution 5

Node Computers 5

Hardware and Software Requirements 6

Node Configuration 6

Node Processes 6

Load Monitoring 6

User 7

User Identification 7

User ID Assignment 7

Enforcement of Privileges 7

User Groups 8

Using EnFuzion 8

EnFuzion Installation and Configuration 8

Executing Runs and Jobs 8

Describing a Run 9

Submitting Runs for Execution 9

Monitoring Run Execution 10

Retrieving the Results 10

Root - Node Communication 10

Starting Nodes 11

Windows NT/2000/XP 11

Linux/Unix 12

Handling of Network Failures 12

Security Issues 13

Submit Environment 13

Directory Layout 13

Executables 13

Configuration Files 13
Root Environment 14
User Account 14
Directory Layout 14
Executables 14
Configuration Files 15
Node Environment 16
User Account 16
Directory Layout 16
Executables 17
Configuration Files 17
Job Execution Environment 18
Handling of Job Execution Errors 19
2. Tutorial 21
Quick EnFuzion Setup Instructions for Windows 21
Obtain Prerequisites 22
Select EnFuzion Hosts 22
Install and Configure One EnFuzion Node 23
Install and Configure the EnFuzion Root 23
Install and Configure One EnFuzion Submit Computer 24
Test the Configuration 24
Add More EnFuzion Nodes 25
Test the Larger Configuration 26
Quick EnFuzion Setup Instructions for Linux/Unix 26
Obtain Prerequisites 28
Select EnFuzion Hosts 28
Install and Configure One EnFuzion Node 28
Install and Configure the EnFuzion Root 29
Install and Configure One EnFuzion Submit Computer 31
Test the Configuration 31
Add More EnFuzion Nodes 32
Test the Larger Configuration 33
Use Your Application with EnFuzion 33
Make a Study Plan 33
Create a Run File 34
Specify Input Files 35
Specify Commands and Output Files 35
Specify Variables 36
Specify Variable Values 36
Prepare Your Application 37
Submit Your Study for Execution 38
3. Windows NT/2000/XP Installation and Operation 41
Installing EnFuzion Software on Windows NT/2000/XP 41
Installing Only EnFuzion Root Software 42
Installing Only EnFuzion Node Software 42
Installing Only EnFuzion Submit Software 42
Reinstalling or Upgrading EnFuzion 42

Installing EnFuzion on Multiple Computers 43
Handling of Installation Problems 43
Installing EnFuzion License 43
Installing EnFuzion Root as a Network Service 44
Network Service Installation 44
The enfstartup Program 45
The enfboot.bat Batch File 46
Starting EnFuzion Nodes at the Computer Boot Time 46
The Setup Program 47
Network Installation on Windows NT/2000/XP 48
The Netsetup Program 49
Netsetup Options 49
Netsetup Commands 49
Remote Installation 50
Windows XP Remote Installation 52
Installation in a Mixed Windows NT/2000/XP and Linux/Unix Environment 52
Modifying the Installation Defaults 52
Removal of EnFuzion Software from Windows NT/2000/XP 53
Windows NT/2000/XP Specific Issues of EnFuzion Operation 53
Starter Service 53
The **service.config** File 54
Remote Commands 54
The Enfkill Utility 55
Performance Considerations 55
4. Linux/Unix Installation and Operation 57
Installing EnFuzion Software on Linux/Unix 57
Installing EnFuzion Root Software 57
Installing EnFuzion Node Software 58
Installing EnFuzion Submit Software 58
Reinstalling or Upgrading EnFuzion 59
Installing EnFuzion on Multiple Computers 59
Handling of Installation Problems 59
Installing EnFuzion License 60
Enabling Linux/Unix Node Computers for EnFuzion Use 60
Configuring EnFuzion Nodes for Remote **ssh** Access 60
Installing EnFuzion Root as a Network Service 61
Network Service Installation on Linux and Mac OS X 61
Manual Network Service Installation 62
Starting EnFuzion Nodes at the Computer Boot Time 63
Installing EnFuzion Node as a Daemon on Linux and Mac OS X 63
Manual Node Daemon Installation 64
Network Installation on Linux/Unix 65
Enfinstall Program 65
Enfinstall Commands 65
Remote Installation 66
Testing Remote EnFuzion Operation 67
Installation in a Mixed Linux/Unix and Windows NT/2000/XP Environment 67

Removal of EnFuzion Software from Linux/Unix 68
Linux/Unix Specific Issues of EnFuzion Operation 68
Performance Considerations 68
5. Submit Configuration 71
Specifying the EnFuzion Service Address 71
The **submit.config** File 71
6. Root Configuration 73
Specifying EnFuzion Node Type 73
The **enfuzion.nodes** File 74
Nodes with Root Control, Connection Initiated by the Root 75
Local Nodes 75
Windows Based Nodes 75
Linux/Unix Based Nodes 77
Access with **ssh** 77
Access with **rsh** 78
Access with **telnet** 78
Custom Node Start 79
Specifying Node Port Number 80
Nodes with No Root Control, Connection Initiated by the Root 80
Direct Nodes 81
Nodes with Root Control, Connection Initiated by the Node 82
WindowsNode Type 82
Nodes with No Root Control, Connection Initiated by the Node 83
Dynamic Nodes 83
Static Nodes 84
Specifying Root Configuration Options 84
The **root.options** File 85
Specifying Available Third Party Software Licenses 85
Enforcing Privileges 85
Rejecting Anonymous Run Submission 86
Prevent Execution of User Programs on the EnFuzion Root System 86
Port Number for the Eye 87
Port Number for the HTTP Based Interface 87
Port Number for Node Connections 87
Port Number for Broadcasting the Address 88
Port Number for Job Execution 88
Port Number for Node Starter Connections 88
Queueing Policy 89
Multiple Remote Nodes from One Host 89
Autonomous Node Operation 89
Wait Limit 90
Deleting Obsolete User Directories 90
Allowing Remote Access to the Dispatcher Interface 91
Restricting Access to the Dispatcher Interface 91
Restricting Access to the HTTP based Interface 92
Restricting Node Access to the Dispatcher 93
Restricting Access to the Eye 94

Starting the Eye 94
Terminating the Eye 95
Off Periods 95
Specifying Mail Server System 95
Specifying Mail Service Port 96
Specifying Mail Sender 96
Concurrent Node Activations 96
Node Restart Period 97
Heartbeat Period 97
Disconnect Period 97
Minimum Time to Obtain Resource Information 98
Complete Logs 98
Maximum Dispatcher Log Size 98
Maximum Datastream Job Size 99
Sample **root.options** File 99
Specifying User Identities 101
The **users** File 102
Specifying Groups 103
The **groups** File 103
Specifying Administrators 104
The **admins** File 104
Specifying User Accounts for Job Execution on Nodes 104
The **user.accounts** File 105
Root Based Security Features 106
Encrypted Passwords in enfuzion.nodes 106
7. Node Configuration 109
Specifying Node User Accounts 109
Specifying Node Configuration Options 109
The **node.config** File 110
Requested Concurrent Jobs 110
Node Port 110
Connect 111
Communication Port 111
Connect Host 112
Connect Port 112
Connect Backup Host 112
Connect Backup Port 113
Connect Retry 113
Connect Delay 113
Execution Time Limit 114
Batch 114
Bind 115
Wait Limit 115
Node Port Message 115
Hello Message 116
Sample **node.config** File 116
Specifying Load Monitoring Options 117

The **enfuzion.options** File....118
System and Local User Options....118
Run Specific Options....118
File Syntax....119
Specifying Time Interval....119
Specifying Days, Months, Date....119
Conditional Options....119
Priority of User Processes....120
Screen Saver....121
Idle Time....122
Temporary Disk Space....122
Working Disk Space....122
Properties....123
Used Virtual Memory Space....123
Stop Virtual Memory Limit....123
Available Main Memory....124
Stop Main Memory Limit....124
Busy Load Limit....124
Stop Load Limit....125
Busy CPU Usage....125
Stop CPU Usage....125
Busy Processor Queue....126
Stop Processor Queue....126
Off and On Periods....126
Stop Processes....127
User Busy Condition....127
User Stop Condition....128
Stop Action....128
Requested Concurrent Jobs....129
Log File Size....129
Log File Fraction....129
Node Directory....130
Termination Signal....130
Mouse Device....130
Console Device....131
Sample enfuzion.options File....131
Specifying Environment Variables....133
The **environment** File....133
Specifying Path Correspondence....134
The **paths** File....134
Specifying Startup Script....135
The **startup.bat** Script....136
Node Based Security Features....136
Trusted Hosts and Executables....136
The **enfuzion.security** File....136
File Syntax....137
Security Considerations in Job Execution Commands....138
User Defined Decryption Primitives....138

Overview of the Dynamic Library 139
Interface 139
Decryption of Passwords 140
Decryption of **enfuzion.security** 140
Library Template 141
Root Authentication 142
The Enfkey Utility 142
Generation and Installation of Keys 143
EnFuzion Provided Authentication Library 143
User Defined Authentication Primitives 144
Overview of the Dynamic Library 144
Interface 144
Returning Status 145
Defining Library Capabilities 145
Displaying Library Information 146
Signing Buffer 146
Verifying Returned Buffer 146
Adding Keys to a Node or a Root Host 146
Removing Keys from a Node 147
Generating New Keys 147
Library Template 147
8. Run Description 153
Introduction 153
Command Line Programs 153
Scripts 154
Parametric Executions 155
Creating a Plan File 156
The Preparator 156
Preparator Wizard 157
Introduction 157
Parameter Description 157
Preprocessing Dialog 159
Input Files Dialog 159
Substitution Files Dialog 159
User Commands Dialog 159
Output Files Dialog 159
Post Processing Dialog 159
Finishing Dialog 159
A Sample Plan 160
Preparing Input Files 160
Initializing the Nodes by Copying Input Files 160
Executing the Jobs 160
Post Processing of Output Files 160
Step by Step Guide through the Wizard 160
Specifying Input Values 164
The Generator 165
A Sample Application Specific Graphical User Interface 165

Description of Plan Files 167
Comments 168
Parameters 168
The Parameter Statement 168
EnFuzion Defined Parameters 171
Tasks 171
The Task Statement 171
Predefined tasks 171
rootstart 171
rootfinish 171
nodestart 172
main 172
onerror 172
Parameter Substitution 172
Locators 173
Task Commands 173
Command **cd** 173
Command **checkfile** 174
Command **checksize** 174
Command **copy** 174
Command **execute** 175
Command **limit** 176
Command **loadparameters** 177
Command **mkdir** 178
Command **onerror** 178
Command **options** 178
Command **server** 179
Command **set** 180
Command **sleep** 181
Command **substitute** 181
Command **updatefile** 182
Command **unset** 182
Conditional Statements 183
Commands from External Scripting Languages 184
Program Enfexecute 185
Configuration Options 185
The Set Statement 186
Including Contents from Other Files 186
Description of Run Files 186
Jobs 186
The Job Statement 187
The Variable Statement 188
Variables 189
Variable Types 189
Options 189
Parameters 189
Scope 189
Retrieving and Setting Values 190

Options 190
Cluster Options 190
Options in **root.options** 191
Node Options 191
Run Options 191
Job Options 193
Context Options 193
Parameters 193
Cluster Parameters 193
Node Parameters 194
Run Parameters 194
Job Parameters 194
Multiple Runs 194
Priorities 194
Run Level 195
Run Weight 195
Preemption 195
Persistence 195
Resource Management 196
Requirements 196
Properties 196
Requirement Matching 196
Timeouts/Error Handling 197
User Errors 197
Timeout for Run Execution 197
Timeout for Job Execution 197
Timeout for User Programs 197
Multiple Job Executions 198
Timeout for Datajob Execution 198
Timeouts for Persistent User Programs 198
Completed Run Directories 198
Datajobs 198
Specifying Datajobs 199
Static Datajobs 199
Streaming Datajobs 199
Datajob Format 200
Executing Datajobs 200
9. Run Execution 201
The Dispatcher 201
The Dispatcher Options 201
Single and Multiple Run Execution 204
Handling of the Eye by the Dispatcher 205
Submitting a Run 205
User Assignment 205
Identification from a Command Line 206
Identification from a Web Browser 206
Identification from a Custom Program 206

Submission from a Web Browser 206
Submission from a Command Line 207
Submitting a Command Line Program 207
Submitting a Script 208
Submitting a Parametric Execution 209
Submission from a Custom Program 210
Submission with the HTTP Based Interface 210
Submission with the EnFuzion API 210
Resubmitting Unfinished Jobs 211
Enfpurge 211
Monitoring Execution 212
Dispatcher Logs 212
The **enfuzion.log** File 212
Description of Log Events 212
Monitoring from a Web Browser 214
Cluster Page 214
Node List Page 214
Single Node Page 214
Run List Page 214
Single Run Page 215
Monitoring from a Command Line 215
Monitoring from a Custom Program 216
Retrieving Results 217
Retrieving Files on the EnFuzion Root System 217
Retrieval with a Web Browser 217
Retrieval from a Command Line 217
Retrieval with a Custom Program 218
Producing Accounting Reports 219
Reports from a Web Browser 219
Reports from a Command Line 219
The **enfreport** Program 220
10. Interfacing with the Dispatcher 223
Graphical Web Based Interface 223
The Eye 223
Using the Eye 223
Submitting a Run 225
Monitoring Execution 227
Cluster Status Page 227
Run List Page 229
Detailed Run Information Page 230
Completed Jobs Page 233
Node List Page 235
Detailed Node Information page 236
Executing Jobs Page 238
Run Results page 240
Used Nodes Page 242
Accounting Page 243

Report Layout Page 244
Report Pages 245
Error Messages List 247
General Error 247
Error: Access Denied 247
Error: Authentication Failed 247
Error: Connection Failed 247
Error: Empty Selection 248
Error: Multiple Selected Items Not Allowed 248
Error: Action Not Permitted! 248
Error: The Eye has Quit 248
Error: Login Failed 248
Error: Dispatcher Not Found 248
Error: No File Name 248
Error: No Such Node 248
Error: No Such Run 248
Error: No Run Results 249
Error: No Reporting Data 249
Error: Page Not Found 249
Error: Session Limit Reached 249
Error: Run Submission Expired 249
Error: Run Submission Failed 249
Error: Mandatory Parameters Missing 249
Error: Invalid Parameter Value 249
Error: Passwords Do Not Match 249
Handling of Privileges 250
Access Control 250
Command Line Interface 250
The Enfsub Program 251
Examples of Using the Enfsub Program 255
The Enfcmd Program 257
Using Enfcmd in a Script 258
Handling of Privileges 259
Access Control 260
HTTP Based Application Programming Interface 260
Description of HTTP Requests 261
Creating a New Run, **POST newrun** 261
Uploading a File, **PUT** 261
Downloading a File, **GET** 261
Deleting a File, **POST deletefile** 262
Checking for File Existence, **POST fileexists** 262
Starting a Run, **POST startrun** 263
Get All Files, **POST getallfiles** 263
Get Input Files, **POST getinputfiles** 263
Get New Files, **POST getnewfiles** 264
Get the Log File, **POST getlogfile** 264
Set a File Copy Mark, **POST setcopymark** 265
Checking for Run Start, **POST runstarted** 265

Checking for Run Completion, **POST runcompleted** 265
Access Control 266
Testing the HTTP Interface 266
Submitting Run for Execution 267
Incremental File Retrieval 267
Implementation of Enfsub in Python 267
Application Programming Interface 268
Connecting with the Dispatcher 268
Format of Messages 269
Error Format 269
Establishing a Connection 269
Direct 269
Observe 270
Description of Commands 270
Cluster Commands 270
cluster get 270
cluster set 270
cluster unset 270
cluster start 270
cluster abort 271
cluster shutdown 271
cluster add run 271
cluster remove run 272
cluster add node 272
cluster remove node 272
Node Commands 273
node get 273
node set 273
node unset 273
node start 273
node terminate 273
Run Commands 274
run get 274
run set 274
run unset 274
run start 274
run stop 274
run abort 275
run approve 275
run reschedule 275
run load 275
run add command 275
run add job 276
run add task 277
run usein datafile 278
run useout datafile 278
run in data 278
run out data 278

run poll data ... 278
run movein datafile ... 279
run copyin datafile ... 279
run moveout datafile ... 279
Job Commands ... 279
job get ... 279
job set ... 279
job unset ... 280
job abort ... 280
job reschedule ... 280
Context Commands ... 280
context set property ... 280
context unset property ... 280
Connection Commands ... 281
connection get ... 281
connection get admin ... 281
connection close ... 281
Handling of Privileges ... 281
Access Control ... 282
Using the Programming Interface From C ... 282
11. Program Reference ... 287
Enfacct ... 287
Enfcmd ... 288
Enfdispatcher ... 289
Enfexecute ... 292
Eye ... 292
Enfgenerator ... 293
Enfinstall ... 294
Enfkey ... 294
Enfkill ... 295
Enfmail ... 295
Enfnodescp ... 297
Enfnodeserver ... 298
Enfpreparator ... 300
Enfprotectpass ... 301
Enfpurge ... 302
Enfreport ... 302
Enfstartup ... 304
Enfsub ... 305
Netsetup ... 309
Setup ... 311
Starter Service ... 312
Uninstall ... 313

A. Frequently Asked Questions 315
1. EnFuzion root programs are not working. How can I proceed? 315
2. An EnFuzion node is not working. How can I proceed? 315
3. The license is not working. How can I proceed? 315
4. Load monitoring is not working. How can I proceed? 316
5. My application is not executing properly on nodes. What should I do? 316
6. Does EnFuzion require Windows NT Server for its operation? 316
7. Does EnFuzion work in mixed Unix and Windows NT/2000/XP networks? 317
8. How can I configure EnFuzion to use Linux/Unix and Windows NT/2000/XP at the same time? 317
9. I am unable to access a Windows NT/2000/XP network drive. 317
10. Can I avoid plain text passwords in the network configuration file **enfuzion.nodes**? 318
11. How can I configure EnFuzion to avoid conflict with a user working on a node? 318
12. How can I configure EnFuzion to execute two simultaneous jobs on a dual processor host? 318
13. How do I manually install EnFuzion on Linux/Unix? 318
14. What are the default installation directories under Unix? 319
15. The installation program on Linux/Unix complains about incorrect user or password on a remote machine. What should I do? 319
16. How does EnFuzion on Linux/Unix communicate with remote machines? 319
17. How does EnFuzion compare to batch queue managers? 319
18. Where can I learn about the early technology behind EnFuzion? 320

Index 321

List of Tables

6-1. Node Types 73
8-1. Available Parameters 190

List of Figures

1-1. How EnFuzion Works 1
2-1. How EnFuzion Works 21
2-2. How EnFuzion Works 26
8-1. Phases of Standard EnFuzion Computation 156
8-2. Preparator Description Dialog 158
8-3. Entering a Preprocessing Command 160
8-4. Entering an Input File 161
8-5. Entering a Parameter Substitution 162
8-6. Entering a User Command 162
8-7. Entering an Output File 163
8-8. Entering a Post Processing command 163
8-9. Sample Output Plan from Preparator 164
8-10. Application Specific Interface in Generator 165
8-11. Interface with All Parameters Defined 166
10-1. The Eye Home Page 224
10-2. The Run Submission Page 225
10-3. Submission of Data Files 226
10-4. Successful Run Submission 226
10-5. The Cluster Status Page 228
10-6. The Run List Page 229
10-7. Detailed Run Information 230
10-8. The Completed Jobs Page 233
10-9. The Node List Page 235
10-10. Detailed Node Information 237
10-11. The Executing Jobs Page 239
10-12. The Run Results page 240
10-13. Run Directory 241
10-14. The Used Nodes Page 242
10-15. The Accounting Page 243
10-16. The Report Layout Page 244
10-17. Run Report 246
10-18. Node Report 246

Preface

EnFuzion turbo charges your applications by harnessing the available CPU power on your computer network and making all computers perform like one big powerful team. EnFuzion users have been able to reduce computational time from months to days, from days to hours and from hours to minutes.

The CPU power can be contributed by any computer on the network, including dedicated or shared servers and standard desktop computers. EnFuzion is ideally suited to exploit the combined power of rack mounted and blade servers. On shared desktop machines, the EnFuzion workload can be made transparent to users.

EnFuzion offers the highest throughput and lowest latency of any resource management product on the market today. It can easily handle thousands of jobs per second with the latency in the sub second range, for example. EnFuzion provides multiuser support and accounting reports, while maintaining security. EnFuzion also supports real time processing through datastream jobs.

Since release 8.0, EnFuzion delivers significant new functionality, including improved ease of use and deployment, fail over capabilities on the root, simplified handling of single jobs, identification of EnFuzion users, increased security, and an expanded range of supported platforms. EnFuzion 8.2 further simplifies the process of submitting the jobs and retrieving the results, implements a new HTTP based interface, and includes an open source program in the Python programming language to demonstrate the use of the new HTTP interface. EnFuzion 9.3 includes improved handling of application errors, so that job execution is fully automated even in most demanding environments. Additionally, resource usage is collected for jobs on Windows and Linux platforms. The release also incorporates a large number of improvements and error fixes while maintaining backward compatibility with previous releases.

The improved web graphical user interface provides an extended set of commands for monitoring and controlling EnFuzion operation from any standard browser. A new set of utilities makes it straightforward to install EnFuzion as a service on the network, so that it can be accessed and used remotely by multiple users. EnFuzion root can be automatically restarted after a machine failure, so that the work lost is minimized. Single jobs can be submitted simply through a command line or a shell script, which removes the need for EnFuzion specific scripts for these jobs. All EnFuzion runs are assigned an owner, which is useful for accounting reports and in allocating access rights for security purposes.

This manual gives you the information you need to deploy EnFuzion across your organization and to exploit EnFuzion's many features.

We discuss the basic concepts in Chapter 1 and provide a short tutorial in Chapter 2. Installation instructions for Windows are given in Chapter 3 and for Linux/Unix in Chapter 4. Configuration is discussed in Chapter 5, Chapter 6 and Chapter 7, presenting submit, root and node configuration, respectively. Description of jobs is presented in Chapter 8. Chapter 9 provides details on job execution. Extensive capabilities to interact with the scheduler are described in Chapter 10. Chapter 11 is a reference chapter, providing details about EnFuzion programs.

We at Axceleon welcome you to EnFuzion 9.3, our approach to extreme clustering and grid computing. We invite you to use EnFuzion to apply the combined CPU power to your computational tasks and do more with your computing infrastructure.

The Worldwide Axceleon EnFuzion Team

January, 2009

Chapter 1. Overview of EnFuzion

The Power of Many

EnFuzion, by Axceleon, is made to harness the power of cluster and grid computing, technologies that connect many distributed computers together to work as one team on a single problem.

EnFuzion makes it easy to execute a large number of jobs over a large number of computers and gain the savings in time and money. It is designed to handle the most complex and demanding computational tasks with minimal overhead. EnFuzion provides facilities to combine the power of hundreds of computers in a single cluster with job throughput rates of several thousand jobs per second. Job duration can range from hours or even days to less than a second, providing results in real time.

EnFuzion handles multiple simultaneous users. It dynamically partitions computing resources, based on job priorities and workloads. EnFuzion provides resource management and job allocation through user defined criteria. The criteria can be based on underlying hardware platforms or available applications, and they can be changed dynamically.

EnFuzion runs on networks ranging in size from a few to several hundred machines. Jobs can be distributed over any TCP/IP based network, whether on the local area network or across the Internet. By using EnFuzion to distribute jobs over multiple computers, it is possible to achieve a processing speed increase of several orders of magnitude. For example, a 10 hour task can be computed in one hour on ten computers. A one month task can be computed in a day on 30 computers. A one year task can be computed over a weekend on 150 computers. And a one day task can be computed in 90 seconds on 1000 computers.

EnFuzion is used in a wide range of applications. The financial services industry, bioinformatics, digital content creation, computer graphics rendering, data mining, operations research, electronic design, and VLSI design are some of examples of its current use.

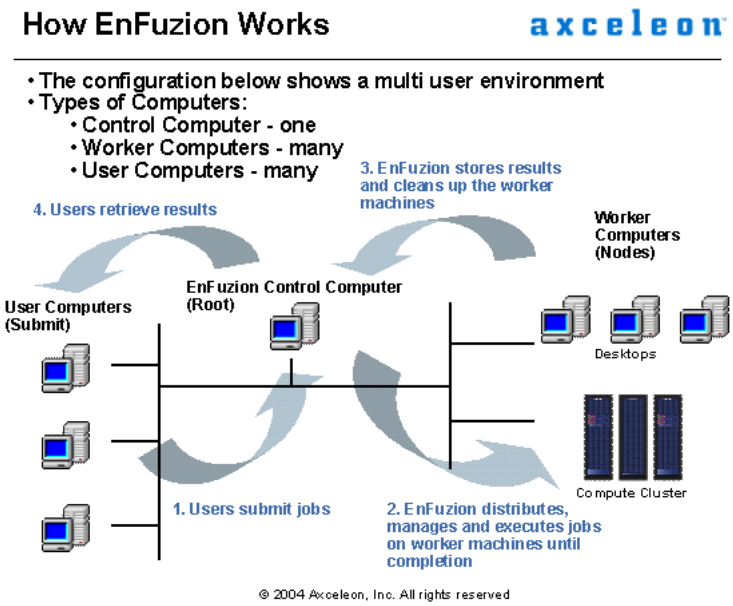
Many applications are ideally suited to run on large computing clusters. Long running applications that perform the same task over and over benefit greatly from the acceleration EnFuzion provides. Take Monte Carlo simulations, for example. Millions of scenarios can be calculated to explore the average or the extreme model behavior for a single application. With EnFuzion it is possible to shorten the calculation time by orders of magnitude or to expand either the number of scenarios investigated or the complexity of the individual scenarios.

Basic EnFuzion Concepts

EnFuzion runs describe work that is scheduled and executed by EnFuzion on remote machines. Runs can be either command line programs, scripts or parametric executions. A parametric execution contains multiple jobs that share execution commands, but have different input parameters and different outputs. Parametric executions are optimized for applications, where the same program is executed again and again, thousands of times if necessary, each time with different input parameters. Normally, each instance of application execution represents one job .

The operation of EnFuzion in a distributed environment is shown in Figure 1-1.

Figure 1-1. How EnFuzion Works


How EnFuzion Works
axceleon
• The configuration below shows a multi user environment
• Types of Computers:
• Control Computer - one
• Worker Computers - many
• User Computers - many
3. EnFuzion stores results and cleans up the worker machines
4. Users retrieve results
Worker Computers (Nodes)
EnFuzion Control Computer (Root)
User Computers (Submit)
Desktops
Compute Cluster
1. Users submit jobs
2. EnFuzion distributes, manages and executes jobs on worker machines until completion
© 2004 Axceleon, Inc. All rights reserved

Runs are submitted by EnFuzion users from submit hosts, which are normally local user machines. Jobs are executed by EnFuzion nodes, which are computer hosts that perform the computation. A central host, called EnFuzion root, controls the nodes and manages job execution.

EnFuzion implements the concept of a user. All interactions with EnFuzion at run time are assigned an owner user ID. User IDs are used for generating activity reports and for restricting permitted actions.

The sections below explain basic EnFuzion concepts in more detail.

Parametric Execution

EnFuzion can manage remote execution of regular command line programs and scripts. Additionally, it is optimized for parametric executions, executing the same application many times with different input parameters. Parametric executions are common in computational modeling, simulations, and analysis. Many tasks can be reduced to parametric executions, such as Monte Carlo analysis, design optimization and verification, computational experiments, data mining, searching, combinatorial optimization, what-if scenarios and other similar tasks.

Each parametric execution is described by a run, which is a container for jobs that perform the same commands with different input values.

Run

User jobs are submitted through runs. Each run specifies an environment for job execution and contains one or more jobs. The number of jobs in a run can range from one to millions of jobs.

A run can be either a command line program, a script or a parametric execution, containing many jobs. A parametric execution consists of tasks , job descriptions and configuration options. Tasks include commands that are executed for each job in the run. These task commands provide instructions on how to execute applications, specify input and output files and such. All jobs in a run share the same tasks. Job descriptions provide specific input values for each job. The run configuration specifies the options that determine run behavior. Run options can determine, for example, run priorities, timeout limits and so forth.

Runs are described in detail in Chapter 8. For run options see the Section called *Options* in Chapter 8.

Job

A job corresponds to one unit of work. It executes commands from the common tasks in the run, but uses its own specific input parameter values.

EnFuzion supports two kinds of jobs. Regular jobs must have an associated task description and a set of input parameters. These regular jobs are simply referred to as jobs. They are used in most applications. Datastream jobs consist of input data and resulting output data. Datastream jobs are referred to as datajobs. They deliver higher throughput, with less overhead than regular jobs and are better suited for certain special applications.

Context

Run execution results in contexts. For each node, executing jobs from a run, the run maintains a context with temporary information about the node. A context is created dynamically during job execution, after a node has been initialized to execute the jobs of the run. An initialization can be installation of the execution binary on the node or the copying of common files to the node. If a run or a node is terminated, the corresponding context is deleted. Contexts are handled automatically by EnFuzion. There is no need for users to issue any special context commands.

Submit Computers

Submit computers are used to submit jobs for execution. These are usually local user machines, although any other machine can be used to submit jobs.

EnFuzion includes programs that allow job submission and retrieval of results from a command line, provide user identification and simplify job preparation.

Users on submit computers can use a standard web browser to submit jobs and communicate with the EnFuzion root. In that case, there is no need to install any EnFuzion related software on the system.

Hardware and Software Requirements

Besides having EnFuzion submit software installed, there are no special requirements for any additional software or hardware on the submit host.

Submit Configuration

If the EnFuzion is used as a service on the network , then the service address must be specified. By default, the address of **localhost:10102** is being used by EnFuzion programs. If the EnFuzion service address is located at a different address, it is specified in the **submit.config** file. Details are provided in the Section called *Specifying the EnFuzion Service Address* in Chapter 5.

Submit Processes

There are no background EnFuzion processes on submit hosts. All EnFuzion processes are executed under the explicit user control.

Job Submission and Retrieval of Results

Users can communicate with the root from their submit computers using a web based interface, a command line program or directly through a network based application programming interface.

Jobs are submitted and results can be retrieved by users from their submit computers through a standard web browser, which communicates with the Eye process on the root. See the Section called *Graphical Web Based Interface* in Chapter 10 for more details on this process.

EnFuzion provides the command line program **enfsub**, which is used to submit jobs and to retrieve results. This command is detailed in the Section called *The Enfsub Program* in Chapter 10. The **enfsub** command is useful to automate job submission in scripts, which can be implemented in standard scripting languages, such as command shells, Perl, Python and Ruby.

EnFuzion provides the command line program **enfcmd**, which is used to monitor and control submission. This command is detailed in the Section called *The Enfcmd Program* in Chapter 10. The **enfcmd** command is useful to automate EnFuzion activity in scripts, which can be implemented in standard scripting languages, such as shells, Perl, Python and Ruby.

Alternatively, other programs in programming languages such as C/C++ and Java can communicate directly with the EnFuzion root through the HTTP based interface, which is optimized for job submission and retrieval of results, or the EnFuzion network based API, which provides a complete range of commands to control the root. See the Section called *HTTP Based Application Programming Interface* in Chapter 10 and the Section called *Application Programming Interface* in Chapter 10 for more details on these two interfaces.

Root Computers

The root is the central component of an EnFuzion cluster. It controls the networked cluster nodes, handles communication with users, and manages the execution of jobs. Each root can control hundreds of nodes and can process thousands, or even millions of jobs, sometimes in just a few minutes.

The root activates and terminates cluster nodes. It exchanges heartbeat messages with nodes to determine their availability. It sends jobs for execution to nodes and retrieves job results.

Hardware and Software Requirements

Besides having EnFuzion root software installed, there are no special requirements for any additional software or hardware on the root host. Since EnFuzion itself introduces little overhead, regular desktop computers can serve as cluster roots, even for very large clusters. In most EnFuzion environments, the load on the root host is light, so almost any computer can act as an EnFuzion root, as long as it provides sufficient disk storage for EnFuzion users.

Root Configuration

Cluster nodes are described in the **enfuzion.nodes** file. For a detailed description see the Section called *Specifying EnFuzion Node Type* in Chapter 6. The root provides a range of user configurable options in the **root.options** file. For a description of root options, see the Section called *Specifying Root Configuration Options* in Chapter 6. The handling of EnFuzion users is configured by several files: **users**, which modifies default user assignments; **groups**, which lists group memberships; **admins**, which specifies users with administrative privileges; and **user.accounts**, which specifies how user accounts are determined on the nodes. These files are described in the Section called *Specifying User Identities* in Chapter 6, the Section called *Specifying Groups* in Chapter 6, the Section called *Specifying Administrators* in Chapter 6, and the Section called *Specifying User Accounts for Job Execution on Nodes* in Chapter 6.

Root Processes

The central process on the root is the Dispatcher, described in Chapter 9. The Dispatcher controls several subprocesses, including the node manager to manage the nodes, the node starter to start the nodes and the job daemon to execute job commands on the root.

The root also hosts the Eye process, which provides a web based user interface to the Dispatcher.

Root Monitoring and Control

The EnFuzion root can be monitored and controlled using any standard web browser, the command line program Enfcmd or directly through a network based application programming interface (API) . Web interface is provided by the Eye, which is executing on the EnFuzion root host. The Eye is described in more detail in the Section called *Graphical Web Based Interface* in Chapter 10. The **enfcmd** command is detailed in the Section called *The Enfcmd Program* in Chapter 10. Finally, the API is described in the Section called *Application Programming Interface* in Chapter 10

Job Execution

When jobs are submitted to the root, the root prioritizes their execution and executes them as nodes become available. The root communicates with nodes in order to maximize job throughput and to assure fast and reliable job execution. Through its resource management capabilities, the root matches job requirements with node capabilities. If a node becomes unavailable or a system error occurs, a job is automatically restarted on one of the working nodes.

Node Computers

Cluster nodes execute user jobs. A cluster can have hundreds of nodes, and each node can be configured to execute more than one user job. Furthermore, more than one cluster node can run on a single computer. This is useful for powerful computers with multiple processors.

Hardware and Software Requirements

Node computers can vary in size and functionality, ranging from desktop computers to powerful servers running Windows NT/2000/XP, Unix, or Linux. There are no special hardware or software requirements for nodes. All that is really required is to have the EnFuzion node software installed and a TCP/IP connection to the root host.

Node Configuration

Nodes provide a range of user configurable options in the **node.config** file , detailed in the Section called *Specifying Node Configuration Options* in Chapter 7. Load monitoring options, which determine when a node is available to execute an EnFuzion job, are specified in the **enfuzion.options** file. The **enfuzion.options** file is detailed in the Section called *Specifying Load Monitoring Options* in Chapter 7.

EnFuzion allows you to specify how file paths on nodes correspond to paths on submit computers. This enables EnFuzion to automatically translate file paths between platforms. The **paths** file provides the path correspondence information (see the Section called *Specifying Path Correspondence* in Chapter 7).

EnFuzion can set the values of environment variables for programs that are executed under its control on the node. The **environment** file configures the environment (see the Section called *Specifying Environment Variables* in Chapter 7).

On Windows only, EnFuzion supports a startup script. This is an optional script, called **startup.bat** and is provided by the user. The script is executed by the EnFuzion node server at the startup and can be used to perform node initialization actions, such as mounting remote file shares (see the Section called *Specifying Startup Script* in Chapter 7).

On Windows only, EnFuzion provides a Starter Service, which starts the local EnFuzion processes on the node. The **service.config** file is a configuration file for the service (see the Section called *The **service.config** File* in Chapter 3).

Node Processes

The main process on the node is the node server. The node server communicates with the root and manages other processes on the node. User jobs are executed by the job server processes. Each job has its own job server processes. The job server manages all aspects of job execution, such as controlling user commands and the copying of files.

Load Monitoring

EnFuzion provides a wide range of load monitoring options for the node hosts. These options specify when a computer is idle and when it is available to execute user jobs. Examples of load monitoring options include ’no interactive use’, ’sufficient available RAM’, ’sufficient available disk space’, and ’low

CPU load'. Options are controlled by system administrators to provide optimal utilization of resources in their computing environment. See the Section called *Specifying Load Monitoring Options* in Chapter 7.

User

EnFuzion implements the concept of a user. All interactions with EnFuzion at run time are assigned an owner user ID. This owner assignment is used in accounting reports to identify the work done by a single user or to restrict user actions.

User Identification

A user is identified by a string in the form **<user>@<host_name>**. By default, **<user>** is the account name of the user that is submitting the run and **<host_name>** is the host name of the computer where the submission is performed. **<host_name>** is usually the fully qualified domain name (FQDN) of the host. If the domain name is not set, but the host name is, it is equal to the host name. Otherwise, it is the IP address of the local host. If EnFuzion is unable to determine the default user ID string, a generic **anonymous** user ID is assigned as the run owner.

The default user ID string can be changed by the EnFuzion administrator through a configuration file on the EnFuzion root system.

User ID Assignment

An EnFuzion user ID is assigned to each run, when the run is submitted for execution. The user assignment cannot be changed later.

To enhance security and simplify usage, EnFuzion delegates the task of user identification to the operating system of the submit computer. When a user connects to EnFuzion for the first time, the user account name on the submit computer and the submit computer host and domain name are used to form a user identification string, which is sent to the Dispatcher on the root system. The EnFuzion user cannot influence the user assignment.

If the run is submitted through a command line, this user identification and assignment are done transparently to the EnFuzion user.

If the run is submitted through a web browser, then the EnFuzion user must perform a login. Otherwise, a generic **anonymous** user ID is assigned as the run owner. The user performs a login by using an identification file that was generated by the EnFuzion **enfcmd** command line utility.

Enforcement of Privileges

An administrator can restrict actions that regular EnFuzion users can perform. By default, there are no restrictions and any EnFuzion user can perform any action.

Privilege enforcement is turned on by the administrator in a configuration file on the EnFuzion root system. This enforcement restricts actions of regular EnFuzion users. They can only add new runs and control runs that they own. They are not allowed to control the cluster by performing actions, such as

removing a run owned by another user, adding and removing nodes, shutting down the cluster, and modifying cluster and node settings and properties.

Even if the privilege enforcement is turned on, there are no restrictions on actions by the users that are identified as EnFuzion administrators. These users are enumerated in a configuration file on the EnFuzion root system.

User Groups

EnFuzion users can be grouped by the administrator in order to report combined activities of related users. Users can be members of one or more user groups.

Groups are useful to generate combined activity reports for different departments or group projects.

Using EnFuzion

The sections below provide an overview of the basic concepts that underlie the execution of jobs by EnFuzion, including software installation and configuration and the handling of runs and jobs.

EnFuzion Installation and Configuration

EnFuzion must be installed on all submit hosts, on the root host and on all participating node hosts . The EnFuzion distribution package includes installation scripts and programs to quickly install EnFuzion on each host type. Since the EnFuzion software components are different for each type, separate installation procedures are provided for submit, root and node hosts. See Chapter 4 and Chapter 3 for details about the installation procedures.

After EnFuzion is installed, it must be configured. As a mandatory configuration step, the submit hosts, the root and the nodes must be configured to be able to find each other and establish a connection.

On submit hosts, the address of the EnFuzion root service must be specified as specified in Chapter 5. Alternatively, if only a web browser is used, the address can be provided explicitly by the user.

Usually, the root is supplied with a list of hosts which are used for the nodes and instructions on how to start and access the nodes. This process is detailed in Chapter 6. An alternative approach, more suitable for dynamic environments where nodes change often, is to configure nodes to contact the root directly and not wait to be started by the root (see the Section called *Nodes with No Root Control, Connection Initiated by the Node* in Chapter 6).

EnFuzion provides a large number of additional configuration parameters, which are available to fine tune EnFuzion behavior for specific user environments. These configuration parameters are optional, and their default values are suitable for most environments. Root configuration is provided in the **root.options** file and in files for dealing with EnFuzion users, detailed in Chapter 6. Node configuration is provided in the **node.config** file, the **enfuzion.options** file, and the **enfuzion.security** file. These files are detailed in Chapter 7.

Executing Runs and Jobs

After the EnFuzion submit hosts, the root and nodes are configured, the cluster is ready to execute user jobs. The process of executing jobs consists of several steps:

- Describing a run
- Submitting the run for execution
- Monitoring the execution
- Retrieving the results

The sections below provide more details about each of these steps.

Describing a Run

A run can be simply a command line program, a script or a parametric execution. In the case of a command line program or a script, there is no need for the user to provide any additional configuration details. A parametric execution contains multiple jobs described with a list of commands to execute, a list of input values for each job and any additional configuration options. EnFuzion provides two different ways to describe parametric executions, either as a plan file or as a run file.

A plan file is a template for the run. It includes descriptions of job parameters, but not their actual values. Plan files are used by EnFuzion to build application specific GUIs, which allow users to quickly generate jobs for parametric executions. A plan file must be converted to a run file, before it can be submitted to the EnFuzion root for execution. Plan files are converted to run files with the EnFuzion Generator program. The Generator is detailed in the Section called *Specifying Input Values* in Chapter 8. Using the plan file, the Generator creates an application specific GUI, which is used by the user to select input values for job parameters and produce a run file.

Plan files are regular text files. EnFuzion also includes the Preparator program, which provides for a simple creation of plan files. The Preparator is detailed in the Section called *The Preparator* in Chapter 8. Alternatively, plan files can be created with standard text editors.

The run file includes a description of the run and input values for job parameters. Run files can be submitted directly to the EnFuzion root for execution.

Run files are regular text files. Depending on the application, they can be produced by using the Preparator and the Generator, by using standard text editors or can be generated by other programs.

Runs are usually prepared on a submit computer, which can be a workstation or a personal computer. The Preparator and the Generator are also executed on the submit computer.

Submitting Runs for Execution

Runs are submitted for execution as a command line, as a script or as a run file. The Dispatcher, which is the controlling EnFuzion process on the root, can be used in a single run mode or in a multiple run mode.

The single run mode is most commonly used interactively. The Dispatcher takes a single run, executes all of the jobs in the run and then exits. In this case, the submit computer and the root computer are the same. Input files and results are provided in the Dispatcher working directory on the submit computer.

The multiple run mode is used to provide EnFuzion as a service on the network. The root computer is usually different from the submit computers, although that is not a requirement. The Dispatcher is able to execute many runs concurrently, even from multiple users. Users submit their run files and their associated input files to the Dispatcher for execution. The submission is done through a web browser on the submit computer (see the Section called *Graphical Web Based Interface* in Chapter 10) or from a command line (see the Section called *The Enfsub Program* in Chapter 10). Another option for submitting runs is from applications, using the HTTP based interface or the EnFuzion network API for direct communication with the Dispatcher.

Monitoring Run Execution

Runs can be monitored with a web browser by connecting to the EnFuzion Eye. The Eye is an intermediary between the user and the Dispatcher. It takes user commands, communicates with the Dispatcher, and generates HTML pages for the browser. The Eye provides monitoring information, such as progress on the run execution and active nodes. This process is detailed in the Section called *Graphical Web Based Interface* in Chapter 10.

EnFuzion also provides the command line program, Enfcmd, which can be used to monitor run execution and the Dispatcher, either from scripts or from a command line. See the Section called *The Enfcmd Program* in Chapter 10 for more details.

EnFuzion also exports a complete monitoring API, which can be used by other applications to communicate directly with the Dispatcher. Using the API is detailed in the Section called *Application Programming Interface* in Chapter 10.

Another method to monitor the Dispatcher is provided through its log . The Dispatcher maintains an extensive log in the file **enfuzion.log**, which is stored in a file in its working directory. This log file is detailed in the Section called *The **enfuzion.log** File* in Chapter 9. If the Dispatcher is executed in a single run mode, the log is also printed on the screen. In addition, each run maintains its own log in its corresponding directory. A run log contains only events that are relevant for that specific run.

Retrieving the Results

The run results are stored in the run subdirectory of the working directory of the Dispatcher on the root computer.

If the Dispatcher is executed in a single run mode, then this directory is on the local computer. The user can access them directly on the local computer.

If the Dispatcher is executed in the multiple run mode, then EnFuzion provides several options to make it simple to copy the files to the local submit computer. The results can be copied to submit computers by using a web browser or the **enfsub** and **enfcmd** programs on the submit computer.

Another option for accessing result files on a remote root computer is to place them on a file system that is shared between the root computer and the submit computers.

Another alternative to access files on the root computer is to use system provided applications, such as **ftp**, **scp** and similar.

Root - Node Communication

EnFuzion's root and node processes communicate by means of standard TCP/IP network protocol. TCP/IP allows EnFuzion to seamlessly combine platforms of different types, such as Linux and Windows, to work on the same problem within a single cluster.

The Dispatcher process on the root host is the central EnFuzion process. It starts and terminates all other EnFuzion processes, including processes on node hosts. If the Dispatcher terminates, all root and node processes are terminated, and all user files on node hosts are deleted. Node processes can be configured, so that they do not terminate with the Dispatcher but are suspended from operation until another instance of the Dispatcher is started. User processes are terminated, and user files are deleted in all cases.

Each node host executes a node process, called a node server, or simply a node. The node maintains a permanent connection with the Dispatcher on the root, exchanges heartbeat information with the root, monitors the load on the local host and handles the execution of all jobs on that node. A single host can run more than one node. Several nodes on a single host are not commonly deployed in production environments, but they can be useful for testing purposes.

The Dispatcher provides facilities to simplify the management of node processes. It is able to handle processes on node hosts in a manner that is transparent to the user. It can start and stop node processes as specified by configuration files or through EnFuzion API commands. Nodes can also be started independently, in which case they initiate the connection with the Dispatcher.

EnFuzion node processes execute under the same local user account as the node server. The account is specified during EnFuzion configuration. This user name can be different for each node and is fully configurable. Linux/Unix EnFuzion nodes can also be configured to execute jobs under user specified accounts instead of under the common EnFuzion account.

The following sections describe the starting of node processes on node hosts and the handling of network errors.

Starting Nodes

EnFuzion provides many mechanisms to start and manage EnFuzion node processes, which makes EnFuzion suitable for a wide range of environments. EnFuzion nodes can be of different types, depending on whether they are started and managed by the Dispatcher or they are started independently of the Dispatcher. EnFuzion node types are described in detail in the Section called *Specifying EnFuzion Node Type* in Chapter 6. This section provides a short overview.

EnFuzion provides several options to handle the starting of EnFuzion node processes by the Dispatcher. In the simplest case, standard methods for remote host access are used to start the nodes. These are described with more detail in the following sections on Windows NT/2000/XP and Linux/Unix. Alternatively, users can completely customize the node starting process by providing a personalized script, instead of using a standard method.

Another option is to start EnFuzion nodes independently of the Dispatcher. These nodes either connect to an already executing Dispatcher, or wait for a connection request from a Dispatcher.

Windows NT/2000/XP

On Windows NT/2000/XP, standard Internet protocols for remote execution are not generally provided.

EnFuzion supplies its own service, called the EnFuzion Starter Service, to start processes on remote nodes. The Starter Service handles initiation of EnFuzion processes on the local host and provides additional system management functionality to the EnFuzion root. See the Section called *Starter Service* in Chapter 3.

Local user access is sufficient to use EnFuzion on Windows. However, administrative rights are required to install EnFuzion on Windows NT/2000/XP.

Linux/Unix

On Linux/Unix, **ssh**, **rsh** and **telnet** are the standard methods to start an EnFuzion node. The use of **ssh** is recommended, since it provides the simplest and the most secure way to start a node.

Besides login access, no other special privileges are required to install and use EnFuzion on Linux/Unix. EnFuzion processes do not require root access and can be run under any user. Running EnFuzion under a regular user strengthens security on nodes, since privileges of EnFuzion processes are limited to privileges of the user under which they execute. An exception is when nodes are configured to allow EnFuzion users to specify node accounts under which their programs are executed.

If the **telnet** protocol is chosen to start a node process, **telnet** access must be enabled on the node.

Although EnFuzion might use the standard **ftp** protocol to speed up the node start process for **telnet**, **ftp** is not required for successful EnFuzion operation. The only exception is remote installation, which uses **ftp** to copy files to node hosts. Alternatively, nodes can be installed without the use of **ftp** by copying the files manually to nodes. After EnFuzion is operational, **ftp** is not necessary.

Handling of Network Failures

EnFuzion detects network failures and provides a wide range of features to deal with them. It handles failed nodes and automatically resubmits any jobs that were executing on a failed or disconnected node to an operational node. There is an exception for nodes that operate in the autonomous mode. In this mode, jobs on disconnected nodes continue with execution and report results when the connection is established. The autonomous mode is turned off by default and must be enabled on the root and on the nodes. For details on the autonomous mode, see the Section called *Autonomous Node Operation* in Chapter 6 and the Section called *Bind* in Chapter 7.

At the basic level, EnFuzion detects when a network connection is disconnected. At this level, EnFuzion relies on the error handling capabilities of the underlying TCP/IP networking protocol. Unfortunately, the protocol capabilities are not sufficient. For example, if the network cable is simply pulled out, it is not detected by the TCP/IP protocol itself, but it must be handled by a higher level.

EnFuzion implements a higher level of error detection through heartbeat between the root and node computers. If a heartbeat is not received within a specified time period, the node is declared down. The heartbeat interval is usually set to several minutes in order to reduce network traffic. Heartbeats work well for jobs that execute for several minutes or more. Short jobs that need a few seconds or less to execute require error detection that is much faster than the one provided by heartbeat.

To handle network failures for very short jobs and to assure maximum throughput for this type of job, EnFuzion provides an additional mechanism, which allows multiple executions of a single job. If a node becomes available for job execution and no other jobs in the run are waiting to be executed, an additional

copy of an already executing job is started. As soon as at least one of the copies completes, other copies are terminated. Users can specify the maximum number of executions of a single job through a predefined run variable, ENFMAX_JOB_COPIES. By default, ENFMAX_JOB_COPIES is set to 1 and only one job copy will execute at any time, i.e., this feature is turned off.

Security Issues

EnFuzion works with security mechanisms provided by the underlying computing platforms. It also includes several enhancements, which strengthen standard system security. These enhancements provide additional security in accessing remote hosts and dealing with sensitive security related information. See the Section called *Root Based Security Features* in Chapter 6 for EnFuzion root based security features and the Section called *Node Based Security Features* in Chapter 7 for EnFuzion node based security features.

Submit Environment

This section gives an overall view of the EnFuzion environment on a submit host. It described the layout of directories on the host, the executables required and EnFuzion configuration files.

Directory Layout

EnFuzion programs on the submit host are executed directly by the user or from scripts. The current directory is used as the working directory. The directory contains user input and output files.

Executables

Submit executables must be in the path accessible to the user. The following executables are provided by EnFuzion distribution packages on the submit host:

- **enfcmd**, the program for communication with the EnFuzion root;
- **enfgenerator**, the program for conversion of plan files to run files;
- **enfpreparator**, the program for generation of plan files;
- **enfsub**, the program for submission of runs for execution;
- **enfsub.py**, a subset of the enfsub program, implemented in Python;
- **enfuzion.py**, a Python library to interface with the HTTP based API.

Configuration Files

The following configuration files are used by EnFuzion on the submit host:

- **submit.config**, the address of the EnFuzion root service.

Root Environment

This section gives an overall view of the EnFuzion environment on a root host. It describes the use of user accounts, the layout of directories on the root, the executables required and EnFuzion configuration files.

User Account

EnFuzion root processes execute under the same user account on the EnFuzion root system. Any user account on the root system can be used to execute EnFuzion root processes. EnFuzion does not impose any requirements on the user account.

If EnFuzion is used locally in a single run mode, then it is usually executed under the local user account.

If EnFuzion is used as a server with many users and executes on a remote system, it is recommended that a special user account is created for the EnFuzion root processes on that system.

Directory Layout

The working directory of the Dispatcher is the main root directory. This directory contains the Dispatcher log file, called **enfuzion.log** , the internal Dispatcher directory **enfinfo** and any additional working files supplied by the user or created during execution. Configuration files can also be placed in this directory.

The **enfinfo** directory contains internal files, which are produced by the Dispatcher and required for its operation. An important subdirectory is **acct**, which contains the data to produce the accounting reports. If this subdirectory is deleted, then the accounting data is lost and reports will be empty.

For each run, the Dispatcher creates a subdirectory in the main root directory, called **run-<run_id>**. This subdirectory contains a run specific log, called **enfuzion-run.log**, and any run specific files, which can be temporary working files, user input files and job output files. The subdirectory also contains files that are required to restart a run.

If the Dispatcher executes in the multi run mode and a run directory is not explicitly deleted by the user within a certain time period after the run completes, the run directory is declared obsolete and is deleted from the Dispatcher working directory. This automatic deletion prevents the accumulation of completed run directories. By default, a directory is declared obsolete 7 days after the run finishes. This default behavior can be changed through the root option ENFCLEANUP_LIMIT as described in the Section called *Deleting Obsolete User Directories* in Chapter 6. The option has no effect in the single run mode.

Executables

Root executables must be in the path accessible to the Dispatcher. The following executables are provided by EnFuzion distribution packages on the root:

- **enfacct**, the shell script that starts **enfacct.bin**;
- **enfacct.bin**, the program for collecting accounting information;
- **enfauth**, a dynamic library used for authentication;
- **enfcmd**, the program that provides a command line user interface to the Dispatcher;
- **enfdispatcher**, the central program;
- **enfecho**, a system independent echo utility for use in tasks;
- **enfexecute**, the program that provides EnFuzion commands in any scripting language;
- **enfeye**, the shell script that starts **enfeye.bin**;
- **enfeye.bin**, the program that provides a web based user interface to the Dispatcher;
- **enfinstall**, the Linux/Unix network installation program;
- **enfjobdaemon**, the program that executes job requests from nodes;
- **enfjobmanager**, the program that executes rootstart and rootfinish tasks;
- **enfkey**, the program for generating public and private keys for root authentication;
- **enfmail**, the program to send a message to an SMTP server;
- **enfnodemanager**, the program that handles communication with nodes;
- **enfnodestarter**, the program that starts nodes;
- **enfprotectpass**, the program for encoding clear text passwords in **enfuzion.nodes**;
- **enfpurge**, the program for removing completed jobs from run files, if the Dispatcher fails unexpectedly;
- **enfreport**, the shell script that starts **enfreport.bin**;
- **enfreport.bin**, the program for generating accounting reports in text;
- **enfrm**, a system independent file deletion utility for use in tasks;
- **libglib***, a dynamic library used by the Eye.

On Windows, some of the programs are part of the **enfdispatcher.exe** executable.

Configuration Files

The following configuration files are used by EnFuzion on the root:

- **admins**, a list of users with administrative rights;
- **enfuzion.nodes**, a description of nodes;

- **enfuzion.pkey**, a root authentication file created by the EnFuzion provided enfauth library;
- **groups**, a list of groups with their members;
- **root.options**, root configuration options;
- **users**, mappings to change default user assignments;
- **user.accounts**, permitted user accounts on nodes.

Node Environment

This section gives an overall view of the EnFuzion environment on a node host. It describes the use of user accounts, the layout of directories on the node, the executables required and EnFuzion configuration files.

User Account

By default, all EnFuzion node processes and user jobs execute under the same user account on the node. This account determines user rights on the system. Any user account on the node system can be used. The accounts can differ between the nodes.

Although EnFuzion does not impose any requirements on the account, it is strongly recommended that the root account is not used for EnFuzion node operation. If possible, it is suggested that a special user account is created and used for installation on all EnFuzion nodes.

The default handling of user accounts on the node can be modified on Linux/Unix nodes. These nodes can be configured to allow users to specify the execution account for the user jobs. Each user can specify his or her own account for job execution. Accounts that can be specified by users can be restricted through a configuration file on the EnFuzion root. To prevent a security risk, EnFuzion node programs do not allow users to specify the system root account, regardless of the configuration on the EnFuzion root.

Directory Layout

EnFuzion creates and maintains a directory hierarchy on nodes, which prevents interference during the concurrent execution of different jobs. Each node creates its own directory. The directory is created in the main EnFuzion directory on the node system. The directory is at the top of the hierarchy for all jobs executed by the node. It contains files specific to the node.

When the Dispatcher connects to a node, a subdirectory is created for the Dispatcher. The Dispatcher directory contains files specific to the Dispatcher.

When a run is started on the node by a nodestart task, a run subdirectory is created within the appropriate Dispatcher subdirectory. This directory is a working directory for the nodestart task. The run subdirectory contains files, specific to this run and usually common to all the jobs. Files with relative file names that have been copied to the node by the nodestart task are located in this directory.

When a job is started by the main task, a job subdirectory is created within the appropriate run subdirectory. This directory is a working directory for the main task. The job directory contains files specific to the job. During the job initialization, files in the parent run directory are made available as local files in the job directory.

After a job completes, its directory on the node is deleted in order to free disk space for other jobs. Similarly, run subdirectories and Dispatcher subdirectories are deleted, when all the jobs from the run complete or the Dispatcher disconnects from the node. This directory deletion prevents any accumulation of obsolete files on the nodes, which significantly reduces the effort that is required to maintain the nodes.

If user jobs are executed under a user specified account and not under the EnFuzion node account, then job directory is set to the home directory of the user specified account. In that case, the job directory is not deleted after the job completes, since this would delete the entire user home directory.

Executables

All node executables must be in the path accessible to the node. The following executables are required by EnFuzion on the node:

- **enfecho**, a system independent echo utility for use in tasks;
- **enfexecute**, allows user jobs to interface with EnFuzion;
- **enfjobserver**, the process that executes a job;
- **enfnodeserver**, the central node process;
- **enfrm**, a system independent file deletion utility for use in tasks.

On Windows, in addition to the executables above, the following files are required:

- **enfstartersvc.exe**, the EnFuzion starter service executable;
- **dbghelp.dll**, a library required by EnFuzion diagnostics;
- **enfauth.dll** and **libeay32.dll**, libraries required for authentication;
- **enfuser.dll**, an optional dynamic library used for decryption.

Configuration Files

The following configuration files are used by EnFuzion on the node:

- **enfuzion.key**, contains root public keys (Windows only);
- **enfuzion.options**, contains load monitoring options;
- **enfuzion.security**, defines trusted hosts and executables;
- **environment**, contains a description of environment variables;

- **node.config**, contains node configuration options;
- **paths**, contains file path translation between different platforms.

The following configuration files are available only on nodes running on Windows:

- **service.config**, the configuration file for the EnFuzion starter service;
- **startup.bat**, contains an optional, user supplied startup script.

Job Execution Environment

When a new job is started on a node, the node creates a job server process, which is responsible for the execution of a single job. The job server interprets and executes the task commands that have been specified for the run. Each job has a separate job server. When a job completes, its job server is terminated.

Executables for user applications that are executed on remote hosts by EnFuzion nodes must be available on these computers and included in the execution search path . If EnFuzion is unable to locate an executable file, the job returns an error. Executables can be either preinstalled or copied as part of the job execution.

Each job executes in its own unique job directory on the node. This prevents interference between files from different jobs. For details on directory handling, see the Section called *Directory Layout* above. All relative file names start from this directory. This unique directory makes it possible to run multiple concurrent jobs on the same computer or on the same shared file system without a conflict between the file names of different jobs. For example, each job can write to a file called output. Although all jobs use the same file name, the files are unique for each job, because they reside in different directories.

The job directory contains files specific to the job. Its parent directory belongs to the run and contains files common to all jobs from that run. During the job initialization, files from its parent run directory are made available as local files in the job directory. If file links are supported, links are established from the job directory to the run directory. On file systems that do not support links, such as some Windows based file systems, files are copied from the run directory to the job directory. The job directory is deleted after the job completes in order to make disk space available for other jobs.

The handling of job directories is different if the job is executing under a user specified account. In that case, the job directory is set to the account home directory, common job files are not copied to the job directory and the directory is not deleted after the job completes. Users are responsible for deleting obsolete files.

A job server may need to execute certain commands on the root host. For that purpose, it maintains a connection with the root host. This connection is separate from the connection between the node and the Dispatcher on the root host. The connection can be either permanent or temporary. A permanent connection is established when the job starts and disconnected when it ends. A temporary connection is established only when commands are issued that require access to the root host. The permanent or

temporary mode of connection can be specified by the user by changing the predefined run variable ENFPERMANENT. The default value is false, which means a temporary connection.

All file references on the root host are relative to the run directory. When a file is copied from a node to the root host, the users usually extend its name with a unique identifier to distinguish files from different jobs. This extension can be constructed from a combination of parameter values or can simply be a system defined parameter, called ENFJOBNAME, which is unique for each job.

Handling of Job Execution Errors

During job execution, EnFuzion distinguishes two types of errors, system errors and user errors.

System errors are caused by computer or network failures. If a node host becomes unavailable or the connection between the root and the node hosts fails, jobs executing on that node are automatically restarted elsewhere. No user intervention is required.

User errors are caused by missing files or user commands that return a non zero exit status. The handling of these errors can be specified with the onerror command. Jobs with user errors can either fail, be repeated on a different computer or continue with the execution, depending on the user specified option.

If a job fails on a node computer, either because the user application fails or one of the EnFuzion commands detects an error, then EnFuzion automatically copies the entire current working directory from the node to the root. The directory is named **error.<job_name>**, where <job_name> is replaced with the name of the job. This ability to view the contents of the remote directory at the time of an error significantly simplifies the problem diagnosis. EnFuzion produces additional files in the **error.<job_name>** directory, which make the problem resolution even easier. These additional files are **stdout**, which contains the standard output, **stderr**, which contains the standard error, and **ENVIRONMENT.txt**, which contains environment variables. This default EnFuzion behavior can be changed by the user provided error handler in the onerror task. Details on the onerror task are provided in the Section called ***onerror*** in Chapter 8.

Job execution errors are described with more detail in the Section called *Timeouts/Error Handling* in Chapter 8.

Chapter 2. Tutorial

This chapter is an introduction to using EnFuzion. It includes a step by step guide on how to use most common EnFuzion features and allows you to quickly become productive by applying EnFuzion to your needs.

The first two sections show how to install EnFuzion and test the configuration in Windows and Linux/Unix environments, respectively. The last section provides guidelines for using your application with EnFuzion.

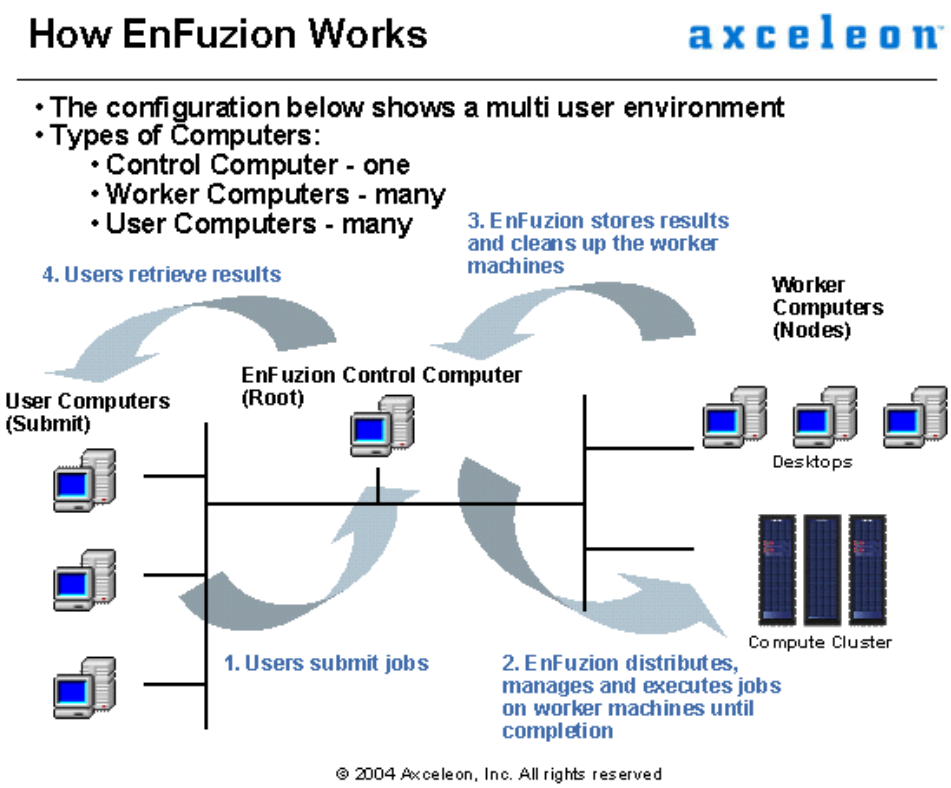
The EnFuzion distribution package includes a sample test study, which is used to test EnFuzion configuration and to illustrate basic concepts of using EnFuzion.

Quick EnFuzion Setup Instructions for Windows

This section describes how to set up EnFuzion for Windows and how to execute the sample test study. If you do not plan to use EnFuzion on Windows, you can skip this section.

The operation of EnFuzion in a distributed environment is shown in Figure 2-1.

Figure 2-1. How EnFuzion Works


How EnFuzion Works
axceleon
• The configuration below shows a multi user environment
• Types of Computers:
• Control Computer - one
• Worker Computers - many
• User Computers - many
3. EnFuzion stores results and cleans up the worker machines
4. Users retrieve results
Worker Computers (Nodes)
EnFuzion Control Computer (Root)
User Computers (Submit)
Desktops
Compute Cluster
1. Users submit jobs
2. EnFuzion distributes, manages and executes jobs on worker machines until completion
© 2004 Axceleon, Inc. All rights reserved

Runs are submitted by EnFuzion users from submit hosts, which are normally local user machines. Jobs are executed by EnFuzion nodes, which are computer hosts that perform the computation. A central host, called EnFuzion root, controls the nodes and manages job execution.

When you are setting up EnFuzion for the first time, start with one EnFuzion node host and expand the configuration with additional EnFuzion nodes only after the initial setup is working. This will give you an opportunity to get familiar with EnFuzion and to resolve any problems early in the installation process.

EnFuzion setup involves the following steps:

- obtain prerequisites;
- select EnFuzion hosts;
- install and configure one EnFuzion node;
- install and configure the EnFuzion root;
- install and configure one EnFuzion submit computer;
- test the configuration;
- add more EnFuzion node computers;
- test the larger configuration.

The steps are described in detail in the sections below.

Obtain Prerequisites

You need an EnFuzion installation package for Windows and an EnFuzion license activation key **enflicense.txt**. The installation package and the license activation key can be obtained from the Axceleon web site at **www.axceleon.com** or by sending an e-mail request to **info@axceleon.com**.

The files from the installation package must be extracted before the install process. If the EnFuzion package is a **.zip** file, then use standard tools to extract the files from the package. If the EnFuzion package is a **.exe**, then this is a self-extracting archive, so just execute the package to extract the files.

The EnFuzion installation package must be available on all machines where EnFuzion is being installed. You can either copy the package to a local disk on each machine or make the package available on a shared network folder.

You need Administrative User Rights to install EnFuzion. These rights are not required for EnFuzion users after the installation.

Select EnFuzion Hosts

Select computers for the EnFuzion root host, one node host and one submit host. The same computer can perform all EnFuzion roles at the same time, so one computer can act as a submit host, an EnFuzion root host and an EnFuzion node host. However, if your planned EnFuzion configuration is large with multiple users, you might not want to have a compute node on the same host as your EnFuzion root.

EnFuzion does not have any special installation requirements for hardware or software, so any Windows NT based system, such as Windows 2000/XP/2003, is suitable. The most important computer is the EnFuzion root. The root controls all EnFuzion activity, so it is important that the host is up and running

continuously. It should also have sufficient disk space to hold user input and output files. Any Windows NT based system can be used for the root, provided that is has enough disk space and is not being turned off regularly.

Install and Configure One EnFuzion Node

EnFuzion node systems are computers that execute jobs. Install EnFuzion on a node as follows:

- login to an account with Administrative User Rights;
- execute **setupnode.exe** from the EnFuzion package;
- create a new account **enfuzion** and set a password for the account.

Install and Configure the EnFuzion Root

The EnFuzion root system is the main computer that controls EnFuzion nodes and job execution. Select your EnFuzion root system and install EnFuzion software as follows:

- login to an account with Administrative User Rights;
- execute **setuproot.exe** from the EnFuzion package;
- use default installation values for directory locations;
- copy the license key **enflicense.txt** to the EnFuzion **config** directory. The default location is **C:\EnFuzion\Config**.
- install the EnFuzion service. Open **Command Prompt** and execute:

  ```
  C:\EnFuzion\Bin\enfstartup install
  ```

  This command installs EnFuzion, so that it starts at the boot time;
- specify the EnFuzion node host in the **enfuzion.nodes** file. The default location for the file is **C:\EnFuzion\Config**. Add the following line to the file:

  ```
  <node_host> enfuzion <password>
  ```

  Replace <node_host> with the name of the node host and <password> with the password for the **enfuzion** account;
- start the EnFuzion service. Open **Command Prompt** and execute:

  ```
  C:\EnFuzion\Bin\enfstartup start
  ```

  This command starts EnFuzion immediately, so that no reboot is required;
- verify EnFuzion operation. Open the Task Manager and confirm that **enfDispatcher** and **enfeye** processes are running. If these processes are not running, check out the EnFuzion log in **C:\EnFuzion\Work\enfuzion.log** for any error messages. If the problem persists, please contact **support@axceleon.com** for assistance;

- verify EnFuzion node operation. Open the following page in your Internet Browser, such as Internet Explorer:

  ```
  http://localhost:10101
  ```

  Follow the **Cluster** link. The **Nodes** table should show 1 Active node. If there are no active nodes, check out the EnFuzion log in **C:\EnFuzion\Work\enfuzion.log** for any error messages. If the problem persists, please contact **support@axceleon.com** for assistance.

Install and Configure One EnFuzion Submit Computer

Submit computers are usually user personal computers. They are used to submit jobs for execution, control and monitor the jobs, and retrieve the results.

The following steps install EnFuzion on submit hosts:

- login to an account with Administrative User Rights;
- execute **setupsubmit.exe** from the EnFuzion package;
- (optional) add the EnFuzion **bin** directory to the **PATH** environment variable. The default location for the directory is **C:\EnFuzion\Bin**.
  - on Windows 2000, go to Control Panel:System:Advanced:Environment Variables;
  - on Windows XP, go to Control Panel:Performance and Maintenance:System:Advanced:Environment Variables;
  - add variable PATH with a value of **C:\EnFuzion\Bin**;
  - reboot the computer.
- specify the EnFuzion root host in the **submit.config** file. The default location for the directory is **C:\EnFuzion\Config**. Add the following line to the file:

  ```
  <root_host>:10102
  ```

  Replace <root_host> with the name of the root host.

Test the Configuration

The EnFuzion package provides a sample application template, which demonstrates EnFuzion use. The template can also be used to test EnFuzion installation. The sample template is installed on the submit computer. It is located in the **C:\EnFuzion\Test** directory by default. If the default is not used, it is in the **test** subdirectory of the user specified EnFuzion directory. The following steps execute the test:

- submit the sample application by double clicking on the **sample.run** file in the EnFuzion **test** subdirectory;
- verify the submission. Open the following page in your Internet Browser, such as Internet Explorer:

```
http://<root_host>:10101
```

Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Runs** link. The Runs table should contain your run, which is called **sample** under the **Name** field and has your user name under the **User** field. The **Run ID** contains a number, which is used by the user to identify the run for results retrieval and other run related operations.

If your run has already completed, then it is moved from the Runs table to the Results table. Its results are available under the **Results** link.

If your run is not in the Runs or in the Results table, then submit the run via a command line, which will provide more details:

- open the Command Prompt;
- go to the EnFuzion **test** directory. The default location is **C:\EnFuzion\Test**.
- submit the sample application:

  ```
  ..\bin\enfsub sample.run
  ```

  Any problems are reported on the screen. If the problem persists, please contact **support@axceleon.com** for assistance;

- obtain the results with the following command:

  ```
  ..\bin\enfsub -attach <run-ID> -rd
  ```

  Replace **<run-ID>** with the run ID of your run, which was obtained during a previous step. This command waits for the run to complete and then copies all its files to a local directory.

Add More EnFuzion Nodes

EnFuzion software must be installed on all systems that will be used as EnFuzion nodes.

- install and configure EnFuzion on each node host as described in the Section called *Install and Configure One EnFuzion Node*;
- add new node hosts to the **enfuzion.nodes** file on the EnFuzion root. The default location for the file is **C:\EnFuzion\Config**. For each new node, add the following line to the file:

  ```
  <node_host> enfuzion <password>
  ```

  Replace <node_host> with the name of the node host and <password> with the password for the **enfuzion** account;
- restart the EnFuzion service. Open **Command Prompt** and execute:

  ```
  C:\EnFuzion\Bin\enfstartup stop
  C:\EnFuzion\Bin\enfstartup start
  ```

These commands restart the EnFuzion service, which is needed to read the new nodes file. Make sure that the EnFuzion service is stopped before it is restarted.

- verify EnFuzion node operation. Open the following page in your Internet Browser, such as Internet Explorer:

  ```
  http://<root_host>:10101
  ```

  Replace <root_host> with the name of the root host. Follow the **Nodes** link. The Nodes table should show the new nodes with Status either Idle or Busy, but not Down.

Test the Larger Configuration

- submit the sample application as described in the Section called *Test the Configuration*;
- verify the submission. Open the following page in your Internet Browser, such as Internet Explorer:

  ```
  http://<root_host>:10101
  ```

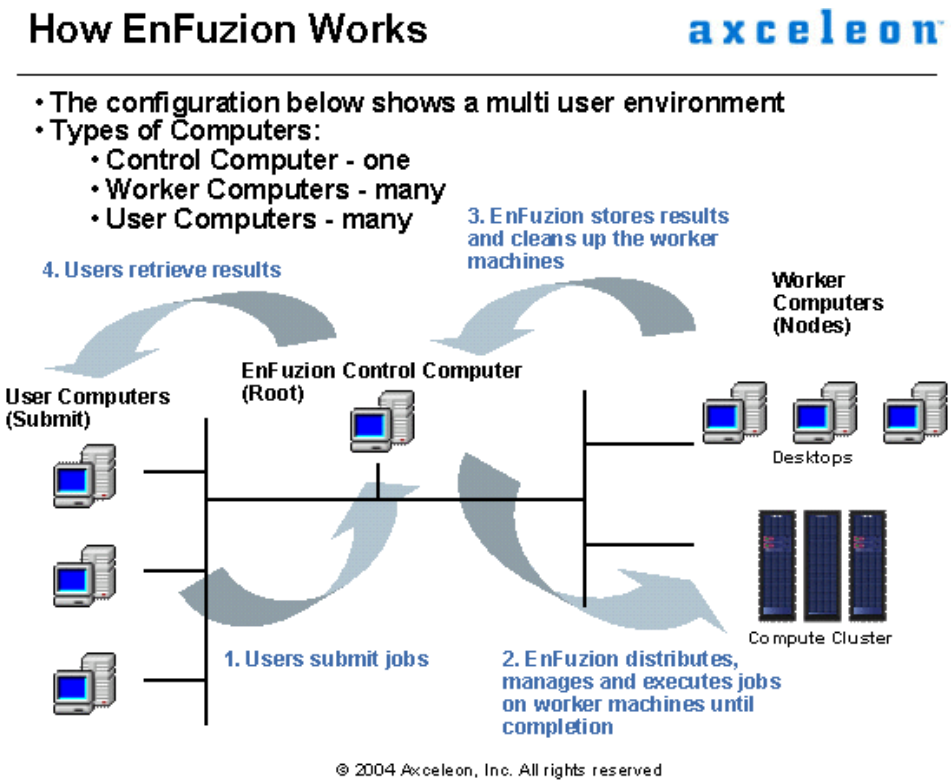
  Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Nodes** link. The new nodes should be executing jobs from the sample test.

Quick EnFuzion Setup Instructions for Linux/Unix

This section describes how to set up EnFuzion for Linux/Unix and how to execute the sample parametric study. If you do not plan to use EnFuzion on Linux/Unix, you can skip this section.

The operation of EnFuzion in a distributed environment is shown in Figure 2-2.

Figure 2-2. How EnFuzion Works


How EnFuzion Works
axceleon
• The configuration below shows a multi user environment
• Types of Computers:
• Control Computer - one
• Worker Computers - many
• User Computers - many
3. EnFuzion stores results and cleans up the worker machines
4. Users retrieve results
Worker Computers (Nodes)
EnFuzion Control Computer (Root)
User Computers (Submit)
Desktops
Compute Cluster
1. Users submit jobs
2. EnFuzion distributes, manages and executes jobs on worker machines until completion
© 2004 Axceleon, Inc. All rights reserved

Runs are submitted by EnFuzion users from submit hosts, which are normally local user machines. Jobs are executed by EnFuzion nodes, which are computer hosts that perform the computation. A central host, called EnFuzion root, controls the nodes and manages job execution.

When you are setting up EnFuzion for the first time, start with one EnFuzion node host and expand the configuration with additional EnFuzion nodes only after the initial setup is working. This will give you an opportunity to get familiar with EnFuzion and to resolve any problems early in the installation process.

EnFuzion setup involves the following steps:

- obtain prerequisites;
- select EnFuzion hosts;
- install and configure one EnFuzion node;
- install and configure the EnFuzion root;
- install and configure one EnFuzion submit computer;
- test the configuration;
- add more EnFuzion node computers;
- test the larger configuration.

These steps are described in the sections below.

Obtain Prerequisites

You need an EnFuzion installation package for your operating systems and an EnFuzion license activation key **enflicense.txt**. Installation packages and the license activation key can be obtained from the Axceleon web site at **www.axceleon.com** or by sending an e-mail request to **info@axceleon.com**.

The files from the installation package must be extracted before the install process. A compressed EnFuzion package has a **.tar.gz** suffix and can be extracted with:

```
tar -zxvf <enfuzion-package>.tar.gz
```

An uncompressed EnFuzion package has a **.tar** suffix and can be extracted with:

```
tar -xvf <enfuzion-package>.tar
```

The EnFuzion installation package for the local operating system must be available on all machines where EnFuzion is being installed. You can either copy the package to a local disk on each machine or make the package available on a shared network directory.

The super user root access is required only for a few limited installation steps. The use of the super user account for all other steps is not recommended. EnFuzion users do not need super user root access.

Select EnFuzion Hosts

Select computers for the EnFuzion root host, one node host and one submit host. The same computer can perform all EnFuzion roles at the same time, so one computer can act as a submit host, an EnFuzion root host and an EnFuzion node host. However, if your planned EnFuzion configuration is large with multiple users, you might not want to have a compute node on the same host as your EnFuzion root.

EnFuzion does not have any special installation requirements for hardware or software, so any Linux/Unix based system is suitable. The most important computer is the EnFuzion root. The root controls all EnFuzion activity, so it is important that the host is up and running continuously. It should also have sufficient disk space to hold user input and output files. Any Linux/Unix based system can be used for the root, provided that is has enough disk space and is not being turned off regularly.

Install and Configure One EnFuzion Node

EnFuzion node systems are computers that execute jobs. Install EnFuzion on a node as follows:

- create a new account **enfuzion** and set a password for the account;
- login to the **enfuzion** account;
- copy an EnFuzion distribution package for your platform to the system and extract the package to a local directory;

- execute the **install-node** script from the EnFuzion package. The script must be executed in its home directory:

  ```
  ./install-node
  ```

Install and Configure the EnFuzion Root

The EnFuzion root system is the main computer that controls EnFuzion nodes and job execution. Select your EnFuzion root system and install EnFuzion software as follows:

- create a new account **enfuzion** and set a password for the account;
- login to the **enfuzion** account;
- copy an EnFuzion distribution package for your platform to the system and extract the package to a local directory;
- execute the **install-root** script from the EnFuzion package. The script must be executed in its home directory:

  ```
  ./install-root
  ```

- add the EnFuzion directory **$HOME/enfuzion/bin** to your PATH environment variable;
- copy the license key **enflicense.txt** to the EnFuzion **config** directory. The default location is **$HOME/enfuzion/config**;
- install the EnFuzion service. This step uses the **install-service** script, which is available with the Linux and Mac OS X EnFuzion packages and has been tested on Red Hat Linux, Suse Linux, Turbolinux and Mac OS X 10.4. For assistance with other platforms, see the Section called *Manual Network Service Installation* in Chapter 4 or contact **support@axceleon.com**.

  Install the EnFuzion service with the following steps:

  - login to the local super user **root** account;
  - execute the **install-service** script from the EnFuzion package. The script must be executed in its home directory:

    ```
    ./install-service
    ```

  - logout from the **root** account. You should be now logged in under the **enfuzion** account;

- specify the EnFuzion node host in the **enfuzion.nodes** file. The default location for the file is **$HOME/enfuzion/config**. Add the following line to the file:

  ```
  <node_host> enfuzion dummy ssh
  ```

  Replace <node_host> with the name of the node host.
- configure **ssh** access from the EnFuzion root to the **enfuzion** account on the node, so that no password is required by the following steps:

- on the root, generate PKI keys, copy the public key to the node and login to the **enfuzion** account on the node:

```
ssh-keygen -d -b 1024
scp ~/.ssh/id_dsa.pub enfuzion@<node_host.node_domain>:
ssh enfuzion@<node_host.node_domain>
```

- on the node, install the public key:

```
mkdir .ssh
chmod 0700 .ssh
cat id_dsa.pub >> ~/.ssh/authorized_keys
chmod 0644 ~/.ssh/authorized_keys
```

- on the root, test the configuration:

```
ssh enfuzion@<node_host.node_domain>
```

  You should be logged in to the node without being asked for a password;

More details on this step can be found at the Section called *Configuring EnFuzion Nodes for Remote **ssh** Access* in Chapter 4.

- start the EnFuzion service. These steps have been tested on Red Hat Linux, Suse Linux, Turbolinux and Mac OS X 10.4. For assistance with other platforms, check the documentation for your operating system or contact **support@axceleon.com**.

  Start the EnFuzion service with the following steps:

  - login to the local super user **root** account;
  - On Linux, start the EnFuzion service:

```
/etc/init.d/enfuzion start
```

    On Mac OS X, start the EnFuzion service:

```
SystemStarter start "EnFuzion Control Root Service"
```

  - logout from the **root** account. You should be now logged in under the **enfuzion** account;

- verify EnFuzion operation. Check processes on the root computer and confirm that **enfdispatcher** and **enfeye** processes are running. If these processes are not running, check out the EnFuzion log in **/var/local/enfuzion/enfuzion.log** on Linux or **/Users/enfuzion/enfuzion/work/enfuzion.log** on Mac OS X for any error messages. If the problem persists, please contact **support@axceleon.com** for assistance;
- verify EnFuzion node operation. Open the following page in your Internet Browser, such as Mozilla:

```
http://localhost:10101
```

Follow the **Cluster** link. The **Nodes** table should show 1 Active node. If there are no active nodes, check out the EnFuzion log in **/var/log/enfuzion/enfuzion.log** on Linux or **/Users/enfuzion/enfuzion/work/enfuzion.log** on Mac OS X for any error messages. If the problem persists, please contact **support@axceleon.com** for assistance.

Install and Configure One EnFuzion Submit Computer

Submit computers are usually user desktop computers. They are used to submit jobs for execution, control and monitor the jobs, and retrieve the results.

The following steps install EnFuzion on submit hosts:

- login to a user account;
- copy an EnFuzion distribution package for your platform to the system and extract the package to a local directory;
- execute the **install-submit** script from the EnFuzion package. The script must be executed in its home directory:

  ```
  ./install-submit
  ```

- add the EnFuzion directory **$HOME/enfuzion/bin** to your PATH environment variable;
- specify the EnFuzion root host in the **submit.config** file. The default location for the directory is **$HOME/enfuzion/bin**. Add the following line to the file:

  ```
  <root_host>:10102
  ```

  Replace <root_host> with the name of the root host.

Test the Configuration

The EnFuzion package provides a sample application template, which demonstrates EnFuzion use. The template can also be used to test EnFuzion installation. The sample template is installed on the submit computer. It is located in the **$HOME/enfuzion/test** directory by default. If the default is not used, it is in the **test** subdirectory of the user specified EnFuzion directory. The following steps execute the test:

- go to the EnFuzion test directory and submit the test:

  ```
  cd $HOME/enfuzion/test
  ../bin/enfsub sample.run
  ```

  Notice the run number that is printed on the screen. It provides a run ID, which is used to monitor the execution and obtain the results;
- verify the submission. Open the following page in your Internet Browser, such as Mozilla:

  ```
  http://<root_host>:10101
  ```

Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Runs** link. The Runs table should contain your run, which is called **sample** under the **Name** field and has your user name under the **User** field. The **Run ID** contains a number, which is used by the user to identify the run for results retrieval and other run related operations.

If your run has already completed, then it is moved from the Runs table to the Results table. Its results are available under the **Results** link.

If your run is not in the Runs or in the Results table, then check out any problems that were reported during the run submission. If the problem persists, please contact **support@axceleon.com** for assistance;

- obtain the results with the following command:

  ```
  ../bin/enfsub -attach <run-ID> -rd
  ```

  Replace **<run-ID>** with the run ID of your run, which was obtained during a previous step. This command waits for the run to complete and then copies all its files to a local directory.

Add More EnFuzion Nodes

EnFuzion software must be installed on all systems that will be used as EnFuzion nodes.

- install and configure EnFuzion on each node host as described in the Section called *Install and Configure One EnFuzion Node*;
- add new node hosts to the **enfuzion.nodes** file on the EnFuzion root. The default location for the file is **$HOME/enfuzion/config** under the enfuzion user. For each new node, add the following line to the file:

  ```
  <node_host> enfuzion dummy ssh
  ```

  Replace <node_host> with the name of the node host.
- enable new EnFuzion nodes for **ssh** access as described in the Section called *Install and Configure the EnFuzion Root*;
- restart the EnFuzion service. Log in to the super user root account on the EnFuzion root host. On Linux, execute:

  ```
  /etc/init.d/enfuzion stop
  /etc/init.d/enfuzion start
  ```

  On Mac OS X, execute:

  ```
  killall enfdispatcher
  killall enfeye.bin
  SystemStarter start "EnFuzion Control Root Service"
  ```

  These commands restart the EnFuzion service, which is needed to read the new nodes file. Make sure that the EnFuzion service is stopped before it is restarted. The commands have been tested on Red Hat Linux, Suse Linux, Turbolinux and Mac OS X 10.4. Consult the documentation for your operating system for other platforms.

- verify EnFuzion node operation. Open the following page in your Internet Browser, such as Mozilla:

  ```
  http://<root_host>:10101
  ```

  Replace <root_host> with the name of the root host. Follow the **Nodes** link. The Nodes table should show the new nodes with Status either Idle or Busy, but not Down.

Test the Larger Configuration

- submit the sample application as described in the Section called *Test the Configuration*;
- verify the submission. Open the following page in your Internet Browser, such as Mozilla:

  ```
  http://<root_host>:10101
  ```

  Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Nodes** link. The new nodes should be executing jobs from the sample test.

Use Your Application with EnFuzion

This section describes how your applications can derive the most benefit from EnFuzion. EnFuzion is optimized for parametric studies, where the same application is executed many times with different input parameters. EnFuzion simplifies parametric studies by automating time consuming, manual steps and provides the results faster by utilizing distributed computing power.

The following steps prepare your application for EnFuzion:

- make a plan for your study, which includes input values, input and output files, and commands to execute your application;
- create a run file, which describes your study. Use values from the previous step;
- prepare your application for parametric studies.

Once a run file is created and the application is prepared, the study can be submitted for execution.

Make a Study Plan

You will need the following information to create a run file for your study:

- **input files**. Identify input files that are required on remote nodes to execute the application;
- **execution commands**. Identify commands that need to be issued on remote nodes to perform the application;

- **output files**. Identify output files that result from the execution on remote nodes and that need to be stored after the application completes;
- **input parameters**. Decide on parameters that your application requires for your parametric study. Decide on parameter names and their values for each execution case.

Create a Run File

A run file describes jobs to be executed for the study. Each job has its own set of values for input parameters, but uses the same input files and executes the same application as other jobs. Each job produces its own results.

A run file must be prepared for each application that performs parametric studies. Usually, the run file is prepared for each application once at the beginning and reused many times.

A run file has the following elements:

- input files. These are provided in the node initialization section. The initialization is executed once on each remote node, before any of the jobs start executing. Input files are shared by all the jobs. Optionally, each job can have its own set of input files;
- job commands and output files. These are provided in the main section. This part is executed once for each job. Job commands are executed with appropriate input values for that job and output results are stored. Job commands are shared by all the jobs. Each job has its own set of output files;
- variables, which provide parameter names. Variables are shared by all the jobs in the study;
- variable values, which provide parameter input values for individual jobs. Each job has its own set of input values.

Run files are regular text files and can be prepared with any text editor or generated by an application. EnFuzion provides additional tools that help with creation of run files (see Chapter 8). A run file template is shown here:

```
task nodestart
    copy <file> node:.
    copy <dir> node:.
endtask

task main
    node:execute <executable> <input_file>
    copy node:<output_file> .
endtask

indexcount 2
variable <var1> index 0 value;
variable <var2> index 1 value;

#jobID    <var1>    <var2>
jobs
```

```
<jobID>   <var1>    <var2>
...
endjobs
```

Individual template elements are described in more detail in the following sections.

Specify Input Files

Input files are specified in the node initialization part in **task nodestart**, which is executed once on each remote node, before any of the jobs start executing.

In node initialization, list all input files that need to be copied to the EnFuzion node. A sample template for node initialization is:

```
task nodestart
    copy <file> node:.
    copy <dir> node:.
endtask
```

Replace <file> and <dir> with a file name or a directory name to be copied. Add more lines if required. When the run file is submitted for execution, make sure that all input files and directories are available in the current directory on the local submit machine.

An example node initialization:

```
task nodestart
    copy input.txt node:.
    copy inputdir node:.
endtask
```

Specify Commands and Output Files

Commands and output files are specified in the **main task**, which is executed once for each job. The task executes job commands with job specific input parameters and stores output files at the end.

In this part of the run file, specify commands that are needed to execute the job and store output files. A sample template for the main task is:

```
task main
    node:execute <executable> <input_file>
    copy node:<output_file> .
endtask
```

Replace <executable> with a path to the application executable, <input_file> with an input file name and <output_file> with an output file name.

An example task main:

```
task main
    node:execute my_program input
    copy node:output .
```

```
endtask
```

An example above has a limitation. All jobs store their results to the file named output. Since each job on a remote EnFuzion node has its own directory, the results are separated initially. However, when they are copied to the EnFuzion root with the copy command, the same name is being used for results from all the jobs, so only the results from the last job are preserved. This limitation can be easily solved by copying the results from each job to a job specific directory on the EnFuzion root.

A revised example task, which copies output files to separate directories, is as follows:

```
task main
    node:execute my_program input
    copy node:output $ENFJOBNAME/.
endtask
```

The task uses EnFuzion provided variable ENFJOBNAME, which gives job ID and is unique for each job. This guarantees that each output file is copied to its own directory and that there is no interference between output files from different jobs.

Specify Variables

Variables are specified in **indexcount** and **variable** statements. Indexcount provides the number of variables. Each variable statement defines one variable, consisting of a variable name and its index location.

A sample template to specify variables is:

```
indexcount <n>
variable <var1> index 0 value;
variable <var2> index 1 value;
```

Replace <var1> and <var2> in the template with variable names for your application. For additional variables, replace indexcount <n> with the total number of variables. For each new variable, add one **variable** statement and increment its index value.

An example variable specification:

```
indexcount 2
variable x index 0 value;
variable y index 1 value;
```

Specify Variable Values

Variable values are specified in the **jobs** section. Each line describes one job and its variable values. The first column is job ID, followed by variable values. Variable with index 0 is in column 2, variable with index 1 is in column 3 and so on. The **jobs** section is ended with **endjobs**.

A sample template to specify variable values:

```
#jobID    <var1>    <var2>
jobs
<jobID>   <var1>    <var2>
...
endjobs
```

Replace <jobID>, <var1> and <var2> in the template with real values. Each <jobID> must be unique. Add more lines for additional jobs.

An example variable specification:

```
#jobID    <var1>    <var2>
jobs
1            1        10
2           11        20
3           21        30
endjobs
```

Prepare Your Application

There is no need to modify your application. However, you need to provide correct values for input variables to the application. These values can be obtained with variable references **$<variable_name>**. Applications usually require input values on the command line or in an input file.

If your application requires an input value on the command line, then simply specify variable reference **$<variable_name>** on the command line. Assume that your application requires one parameter, which provides a year. An example program execution in the run file would look like:

```
node:execute my_program $year
```

In this case, **year** is a variable that must be defined with a **variable** statement, which is described in the Section called *Specify Variables*. Before the job starts executing, EnFuzion automatically replaces **$year** with a variable value for that job.

If an input value is required in an input file, then replace specific values in the file with variable references. Assume that the application requires two values, a year and a month. Before modifying the input file for EnFuzion, it looks like:

```
2003 11
```

To prepare the application for parametric studies, the concrete values for the year and the month are replaced with variable references:

```
$year $month
```

In this case, **year** and **month** are variables that must be defined with a **variable** statement, which is described in the Section called *Specify Variables*. Additionally, variable references in the input file must be explicitly replaced with values with the EnFuzion **substitute** command:

```
node:substitute modified-input.txt input.txt
```

This command takes file **modified-input.txt**, replaces all variable references with their values for a specific job and produces **input.txt** as a result. The command must be executed before **input.txt** is used by other commands. In this example, **modified-input.txt** is prepared before the run is submitted and should be copied to the node as part of the node initialization, which is described in the Section called *Specify Input Files*. The **input.txt** file is not required during node initialization, since it is generated by EnFuzion during the execution.

Variable references can be used in all EnFuzion commands in a run file. In addition to providing input values, they are also useful for naming input and output files. For example, the results from each job can be copied to a different file with the following command:

```
copy node:output $ENFJOBNAME/.
```

$ENFJOBNAME is an EnFuzion variable that provides job ID. In the example above, each job stores results in its own directory. The name of the directory is the job ID.

Submit Your Study for Execution

Once the run file is created and the application is prepared, you are ready to submit the study for execution. It is assumed that EnFuzion has been installed and is operational as described in the Section called *Quick EnFuzion Setup Instructions for Windows* and the Section called *Quick EnFuzion Setup Instructions for Linux/Unix*.

On Windows, the study is submitted as follows:

- submit your application by double clicking on your run file;
- verify the submission. Open the following page in your Internet Browser, such as Internet Explorer:

  ```
  http://<root_host>:10101
  ```

  Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Runs** link. The Runs table should contain your run, which is called **sample** under the **Name** field and has your user name under the **User** field. The **Run ID** contains a number, which is used by the user to identify the run for results retrieval and other run related operations.

  If your run has already completed, then it is moved from the Runs table to the Results table. Its results are available under the **Results** link.

- copy the results to your local host with the following command:

  ```
  ..\bin\enfsub -attach <run-ID> -rd
  ```

  Replace **<run-ID>** with the run ID of your run, which was obtained during a previous step. This command waits for the run to complete and then copies all its files to a local directory.

On Linux/Unix, the study is submitted as follows:

- go to the directory with your run file and input files and submit the test:

  ```
  cd <your_directory>
  ```

```
$HOME/enfuzion/bin/enfsub <your_run>.run
```

Notice the run number that is printed on the screen. It provides a run ID, which is used to monitor the execution and obtain the results;

- verify the submission. Open the following page in your Internet Browser, such as Mozilla:

  ```
  http://<root_host>:10101
  ```

  Replace **<root_host>** with the host name of the EnFuzion root system. Follow the **Runs** link. The Runs table should contain your run, which is called **sample** under the **Name** field and has your user name under the **User** field. The **Run ID** contains a number, which is used by the user to identify the run for results retrieval and other run related operations.

  If your run has already completed, then it is moved from the Runs table to the Results table. Its results are available under the **Results** link.

- obtain the results with the following command:

  ```
  $HOME/enfuzion/bin/enfsub -attach <run-ID> -rd
  ```

  Replace **<run-ID>** with the run ID of your run, which was obtained during a previous step. This command waits for the run to complete and then copies all its files to a local directory.

Chapter 3. Windows NT/2000/XP Installation and Operation

This chapter explains how to install and operate EnFuzion software on Windows NT/2000/XP computers. The EnFuzion software consists of the EnFuzion root components, the EnFuzion node components and the EnFuzion submit components. These components must be installed on computers that will act as EnFuzion roots, EnFuzion nodes or EnFuzion submit hosts, respectively. This chapter covers EnFuzion software installation, EnFuzion license installation, installation of EnFuzion as a network service, network installation, installation in a mixed Windows NT/2000/XP and Linux/Unix environment, instructions on how to modify installation defaults, removal of EnFuzion software and Windows specific issues of EnFuzion operation.

This chapter describes the installation of the EnFuzion package, which contains a text installer. EnFuzion can be also obtained in a package that contains a graphical installer. The installation of EnFuzion with the graphical installer is described in a separate document.

Installing EnFuzion Software on Windows NT/2000/XP

EnFuzion software must be installed on each Windows NT/2000/XP computer to be used as an EnFuzion root, an EnFuzion node or an EnFuzion submit host. The simplest method to install EnFuzion on Windows NT/2000/XP is by executing the **setup** program, which is included in the distribution package. The **setup** program automatically installs all EnFuzion components, including the EnFuzion root, the EnFuzion node and the EnFuzion submit software.

To install EnFuzion software, perform the following steps:

- Obtain the EnFuzion distribution package for Windows NT/2000/XP. Packages are available from the Axceleon (http://www.axceleon.com) web site.
- Unpack the package. If the package is a self-extracting executable with the **.exe** suffix, then simply execute the package by clicking on the file. The package execution will extract EnFuzion distribution files to a folder with the same name as the original package, but without the **.exe** suffix. If the package is an archive with the **.zip** suffix, then unpack the package to a temporary directory. The distribution package and the extraction directories can be deleted after the installation, since they are not required for EnFuzion operation.
- Check installation prerequisites on your Windows NT/2000/XP system:
  - If you use Windows NT, Service Pack 6 is recommended for use with EnFuzion.
  - On some older versions of Windows, TCP/IP protocol is not installed by default. TCP/IP is required by EnFuzion for communication between the root and the nodes. If TCP/IP is not installed on your computer, check Control Panel:Network:Protocols and install TCP/IP.
  - Obtain Administrative User Rights on the system by logging in under the administrator account or under another account with Administrative User Rights. These administrative rights are required only for the installation, they are not required for regular EnFuzion use.

- Install EnFuzion by executing the **setup** program in the directory with extracted EnFuzion distribution files. The **setup** program asks for the EnFuzion installation directory and for the EnFuzion temporary directory. The use of default values is recommended. The default EnFuzion directory is **C:\enfuzion**. The default EnFuzion temporary directory is **C:\enfuzion\temp**.
- (optional) Add the path to EnFuzion executables to the PATH environment variable. This step allows you to execute EnFuzion binaries from a command line without specifying the entire path. The default path for EnFuzion executables is **C:\enfuzion\bin**. If the default EnFuzion value for the EnFuzion installation directory is changed, executables are located in the **bin** subdirectory.

> **Note:** The **setup** program installs a Windows system service, called Starter Service. The service enables a remote EnFuzion root to start jobs on the computer. If the service is not available on a Windows node, EnFuzion will be unable to use that node. The **setup** program automatically configures the service, so that it is always available. No special configuration steps are required.

Installing Only EnFuzion Root Software

The distribution package provides the **setuproot.exe** program for installing only EnFuzion root software on a system. The program can be executed instead of the **setup** program to perform the root only installation.

> **Note:** If **setuproot.exe** is used on a system that already has EnFuzion installed with **setup.exe** or **setupnode.exe**, the installation must be first removed with **uninstall**. **uninstall** is not required if **setuproot.exe** was used for the previous EnFuzion installation.

Installing Only EnFuzion Node Software

The distribution package provides the **setupnode.exe** program for installing only EnFuzion node software on a system. The program can be executed instead of the **setup** program to perform the node only installation.

Installing Only EnFuzion Submit Software

The distribution package provides the **setupsubmit.exe** program for installing only EnFuzion submit software on a system. The program can be executed instead of the **setup** program to perform the submit only installation.

Reinstalling or Upgrading EnFuzion

If EnFuzion is already installed on the system, you can simply repeat the installation process. The **setup** will automatically use the existing EnFuzion installation directories without asking for new values. This

features simplifies the EnFuzion upgrade process. If you wish to change EnFuzion installation directories, uninstall EnFuzion before executing the **setup** program.

The installation process will keep the existing configuration files, but it will upgrade all other files. If a previous configuration file already exists, the new file will be copied to the target directory with the **.new** suffix added to its name.

Make sure that you upgrade EnFuzion root, node, and submit software at the same time, using the same EnFuzion release.

Installing EnFuzion on Multiple Computers

When EnFuzion is being installed to multiple computers, the distribution package is copied to each system, unpacked, and then installed. This installation process can be accelerated by unpacking the distribution package on one system only, and then sharing the directory with the extracted EnFuzion distribution files. This makes the EnFuzion files accessible from other computers, so there is no need for any further file copying and unpacking. In this case, EnFuzion is installed simply by executing the **setup** program on each computer. For an automated installation of EnFuzion across multiple computers, refer to the Section called *Network Installation on Windows NT/2000/XP* below.

Handling of Installation Problems

If you experienced any problems during the installation process, you can send e-mail to support@axceleon.com to report the problems. Include the following information:

- output from the failed installation process
- optionally, a description of your system generated with command **WINMSD /a /f** from computers with a failed installation. This command generates file **<hostname>.txt** with detailed information about the system.

Installing EnFuzion License

EnFuzion software will not work without a license file being installed on the root system. EnFuzion node and submit computers do not require a license. To install an EnFuzion license file on the system, rename the file with an EnFuzion license to **enflicense** or **enflicense.txt** and copy the file to the **config** subdirectory of the EnFuzion installation directory. The default path for the **config** subdirectory is **C:\enfuzion\config**.

The **setup** program can also be used to install an EnFuzion license. If the program finds a file named **enflicense** or **enflicense.txt** in the distribution directory, it installs the license. This capability is useful when installing EnFuzion on a large number of systems, since it automates the license installation step. The license file is simply placed in the unpacked distribution directory before the installation process. The **setup** program then automatically installs the license while installing other EnFuzion components.

EnFuzion licenses can be purchased from Axceleon. Please contact Axceleon or send an e-mail to sales@axceleon.com for details.

Evaluation EnFuzion licenses are available from the Axceleon (http://www.axceleon.com) Web site.

Installing EnFuzion Root as a Network Service

The Dispatcher can be installed as a network service, which means that it automatically started at the computer boot time and available to remote users over the network. This configuration is suitable for environments where one Dispatcher is used by multiple users, and jobs are submitted remotely from the user computers.

EnFuzion provides two programs for a straightforward network service installation and management on Windows computers. The **enfstartup** program installs, uninstalls, starts and stops the EnFuzion network service on the system. The **enfboot.bat** is the batch file that is executed by the system at the boot time to start the Dispatcher, which provides the network service.

This section first provides installation instructions and then describes the **enfstartup** and **enfboot.bat** programs in more details.

Network Service Installation

To install EnFuzion network service, perform the following steps:

- Install EnFuzion on the system as described in the Section called *Installing EnFuzion Software on Windows NT/2000/XP*.
- Install the service with the following command:

  enfstartup install

  This command registers the batch file **enfboot.bat** to execute at the system boot time. This will make the EnFuzion Dispatcher available for job submission on port 10102 and the Eye on port 10101 after the system is rebooted.

  After the EnFuzion service is installed with the **enfstartup install** command, the Windows system configuration needs to be enabled to activate the command at the Windows startup time. This configuration step needs to be executed on the system only once. Perform the following steps:

  - Start the Microsoft Management Console. In the **Start** menu, select **Run...** and enter **mmc**.
  - Add **Group Policy Snap-In**:
    - In **Console/File** menu, select **Add/Remove Snap-In...**;
    - Click **Add...**;
    - Select **Group Policy**;
    - Click **Add**;
    - Click **Finish**;

- Click **Close**;
- Click **OK**;

- Change the Windows startup configuration to activate the startup procedure:
  - Double click **Local Computer Policy**;
  - Double click **Computer Configuration**;
  - Double click **Windows Settings**;
  - Click **Scripts (Startup/Shutdown)**;
  - In the panel on the right, double click **Startup**;
  - Click **OK**;
  - Exit the **Microsoft Management Console** program.

- (optional) Start the Dispatcher manually with the command:

  enfstartup start

  This command starts the Dispatcher immediately, which avoids the need to reboot the system. If the Dispatcher is already running, then this command has no effect. To restart the Dispatcher, use **enfstartup stop** first, followed by **enfstartup start**.

The enfstartup Program

The EnFuzion **enfstartup** program simplifies service installation on a Windows computer. It provides service installation, uninstallation, start and stop.

By default, it uses the EnFuzion provided batch file, which is located in **config\enfboot.bat**. Although the Dispatcher provides a network service, it is executing as a regular program, and not as a Windows service program. The Dispatcher is running under the System account.

The **enfstartup** program takes the following command line arguments:

```
enfstartup \
    install [<startup_script>]
    uninstall [<startup_script>]
    start [<startup_script>]
    stop
```

- **install [<startup_script>]**

The batch file in **<startup_script>** is registered with Windows to execute at the boot time. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory. Make sure that Windows is configured for starting programs at the boot time as described in the Section called *Network Service Installation*.

- **uninstall [<startup_script>]**

  The batch file in **<startup_script>** is removed from files to execute at the Windows boot time. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory.

- **start [<startup_script>]**

  The batch file in **<startup_script>** is executed immediately. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory. If the Dispatcher is already running, then this command has no effect. To restart the Dispatcher, use **enfstartup stop** first, followed by **enfstartup start**.

- **stop**

  The EnFuzion root processes on the system are terminated.

The enfboot.bat Batch File

The **enfboot.bat** batch file starts the EnFuzion service on the system. The file is located in the **config** EnFuzion directory.

Default values are 10102 for the service port and **C:\enfuzion\work** for the Dispatcher main directory. The Dispatcher will restart any uncompleted runs from a previous Dispatcher instance.

Modify the **enfboot.bat** file to change any default values.

Starting EnFuzion Nodes at the Computer Boot Time

EnFuzion nodes can be configured to start automatically at the computer boot time. These nodes can then connect to an EnFuzion root or wait to be contacted, depending on the root and node configuration (see the Section called *Specifying EnFuzion Node Type* in Chapter 6). This automated node start is especially suitable for highly flexible environments, where node or even root computers change often. It simplifies EnFuzion configuration by eliminating the need for the **enfuzion.nodes** file.

Nodes are started by the EnFuzion Starter Service, which must be configured appropriately. The EnFuzion Starter Service configuration file is called **service.config** and is located in the main EnFuzion installation directory. The default location is **C:\enfuzion\service.config**.

To start an EnFuzion node at the computer boot time, add the following line to the **service.config** file:

```
node <user_account> <password> <args>
```

Replace **<user_account>** and **<password>** with the user name and its corresponding password for the user that is used to execute EnFuzion node processes. **<user_account>** and **<password>** can be encrypted as described in the Section called *Encrypted Passwords in enfuzion.nodes* in Chapter 6.

<args> are command line arguments for the node server, described in the Section called *Enfnodeserver* in Chapter 11. These arguments can be used to determine if the node connects to the EnFuzion root or if the node waits for a root connection. For details on connecting a node to the root, see the Section called *Nodes with No Root Control, Connection Initiated by the Node* in Chapter 6. For configuration details on connecting the root to a node, see the Section called *Nodes with No Root Control, Connection Initiated by the Root* in Chapter 6.

After the configuration file **service.config** is modified, the EnFuzion Starter Service must be restarted for changes to take the effect. This restart can be accomplished by rebooting the computer or by manually restarting the service.

If the node is configured to connect to the EnFuzion root instead of the root connecting to the node, then the root must be enabled for this feature to work. By default, connections from external nodes are rejected by the EnFuzion root and nodes will fail to connect. The **rootport** root option, which is described in the Section called *Port Number for Node Connections* in Chapter 6, enables the EnFuzion root for external node connections. Make sure that the **rootport** option is configured on the EnFuzion root, before configuring nodes to connect to the root.

Example:

```
node enfuzion enfuzion -b -d -n 0 0
```

The node automatically detects an EnFuzion root on the same network and connects to it.

Example:

```
node enfuzion enfuzion -b -d -n 192.168.0.1 10103
```

This example is similar to the example above, except that the node connects to a specific EnFuzion root at host 192.168.0.1 and port 10103.

The Setup Program

The Windows EnFuzion installation and upgrade program is called **setup**. Most often, the user executes the program by clicking on the file. In that case, **setup** asks for any user options and installs EnFuzion software on the system. The program also provides additional command line options, which are useful for remote and automated management. This section provides details on the **setup** options.

The **setup** program takes the following command line:

```
setup [ <options> ]
```

- **-main <directory>**

  Define the main EnFuzion directory. Default value is **C:\enfuzion**. If EnFuzion is already installed on the system, this option has no effect.

- **-tmp <directory>**

  Define the EnFuzion temporary directory. Default value is **C:\enfuzion\temp**. If EnFuzion is already installed on the system, this option has no effect.

- **-node**

  Install only EnFuzion node components.

- **-root**

  Install only EnFuzion root components.

- **-submit**

  Install only EnFuzion submit components.

- **-force**

  Force program installation. By default, **setup** does not overwrite an executable file, if it is being used by a process. With this option, the program is terminated, so that the installation of the file can be completed successfully.

- **-noprompt**

  Use default values. With this option, **setup** does not request any input from the user. The program uses default values to perform the installation.

- **-s**

  Perform a silent installation. Do not produce any output and do not request any input from the user.

- **-ignore**

  Ignore errors during program installation. This option is applicable for EnFuzion upgrades. If an EnFuzion program is executing during an upgrade and the **-s** option is turned on, the **setup** program terminates by default. With this option, any errors while upgrading executing programs are ignored and the upgrade proceeds.

Network Installation on Windows NT/2000/XP

The EnFuzion distribution package provides the program netsetup, which is able to install EnFuzion on remote Windows NT/2000/XP hosts from a central location.

The Netsetup Program

The Netsetup program can be used to install EnFuzion on remote systems, without any need to access the system's keyboard or monitor. The program can also be used to control the EnFuzion Starter Service on remote computers. Netsetup is implemented only on the Windows NT/2000/XP platform. On Linux/Unix, the enfinstall program provides similar functionality, as described in Chapter 4.

The **netsetup** program is called with a set of options, followed by a command and possible command options:

```
netsetup [ <option> ] <command>
```

Netsetup Options

- **-v**

  Prints the **netsetup** program version and options.

- **-d**

  Reads EnFuzion nodes from standard input instead of from the file **install.nodes**.

- **-p**

  Prints command progress.

- **-t <number>**

  Executes the command concurrently on at most <number> hosts. The default value is 1, so the command is executed sequentially for each host.

Netsetup Commands

- **install \\<host>\<share>\<source> <destination>**

  Installs EnFuzion executables from a source directory to the destination directory on hosts specified in the file **install.nodes**. Options are as follows:

- **<host>** is the name of the host where the EnFuzion package has been unpacked and has been made available for access over the network.
- **<share>** is the name of the share on the <host>, which contains the <source> directory.
- **<source>** is the directory containing the **setup** program and other EnFuzion distribution files.
- **<destination>** is required for the initial EnFuzion installation. It specifies the EnFuzion installation directory. Its recommended value is **C:\enfuzion**. <destination> is not required, if EnFuzion is already installed on systems.

- **uninstall**

  uninstalls EnFuzion from all hosts.

- **start**

  starts the EnFuzion Starter Service.

- **stop**

  stops the EnFuzion Starter Service.

- **delete**

  deletes the EnFuzion Starter Service from the service control manager database.

- **verify**

  prints EnFuzion Starter Service status information.

Remote Installation

The following steps are required to perform a remote installation:

- Unpack the EnFuzion distribution package into the source directory.
- Make the source directory a shared directory so that it is visible to other system where EnFuzion is to be installed. On a Windows XP source computer, simple file sharing must be disabled (see the Section called *Windows XP Remote Installation*).
- Manually install EnFuzion on the local system from the source directory to the local destination directory.

- Create the file **install.nodes** containing all the nodes where EnFuzion is to be installed. Place the **install.nodes** file in the **Config** subdirectory of the main EnFuzion directory. The default path is **C:\enfuzion\config**.

  For the initial EnFuzion installation, the EnFuzion Starter Service is installed on all EnFuzion node computers. Users can specify domain and user name which are used to make initial installation of the EnFuzion Starter Service. User and domain name can be specified in the **install.nodes** file by writing the user name as **domain_name\user_name** or as **user_name@domain_name**. In case the user name is written without a domain, the user name on the local host is used for the initial Starter Service installation, which is equivalent to **.\user_name**. The node user must have the rights to connect to the node from the network, to install a service on the node system and to connect back from the node system to the system with the installation directory share.

  To make an initial installation of EnFuzion, all the users on nodes in file **install.nodes** must have administrative rights.

- Grant required user rights, if the installation is performed for the first time. They are necessary to install and start the Starter Service the first time. The additional user rights must be granted to the user, usually administrator, that is used to run the netsetup command. These rights must be granted on the root system and on all node systems. In addition to default user rights set when Windows NT/2000/XP are installed, the following additional user rights must be set for a user which will be used for initial EnFuzion installation:

  - Act as part of the operating system
  - Log on as a service
  - Replace a process level token

    If EnFuzion has already been installed on a node and the current installation is performed by a user without administrative rights, all EnFuzion programs except the Starter Service will be upgraded.

- Run the Netsetup program.

  Netsetup executes the **setup.exe** program on all nodes where EnFuzion is installed. For example, to install EnFuzion from host gemini containing EnFuzion files on drive **D:\enFuzion\install** with drive D: shared as "d", execute the command:

  netsetup install \\gemini\d\enfuzion\install C:\enfuzion

  In case EnFuzion security features are enabled on nodes, the host from which netsetup is executed and the **setup.exe** must be defined as trusted in file **enfuzion.security**. A full path name of **setup.exe** must be specified in the security file:

  \\nthost\share\source_dir\setup.exe

  To be able to perform an installation in the example above, the following lines must be added to the **enfuzion.security** file on nodes:

  allow host gemini

  allow executable \\gemini\d\enfuzion\install\setup.exe

Windows XP Remote Installation

In order to remotely install and start the Starter Service on a Windows XP computer, simple file sharing must be disabled on the root system.

Follow the following steps to disable Simple File Sharing on XP Professional:

- Click Start > My Computer > Tools > Folder Options
- Select the View tab
- Go to Advanced Settings,
- Clear the Use Simple File Sharing box
- Click Apply

Installation in a Mixed Windows NT/2000/XP and Linux/Unix Environment

The **setup.exe** works only on Windows NT/2000/XP hosts. It does not support EnFuzion installation on Linux/Unix hosts. A separate installation program is provided for Linux/Unix hosts. See Chapter 4.

Perform the following steps to install EnFuzion in a mixed Windows NT/2000/XP and Linux/Unix environment:

- Install EnFuzion on all Windows NT/2000/XP hosts
- Install EnFuzion on all Linux/Unix hosts
- Include all hosts in your configuration file **enfuzion.nodes**
- If your root computer is a Windows NT/2000/XP host, enable **telnet** access to Linux/Unix hosts, and add the string "Unix" to all Linux/Unix hosts in the **enfuzion.nodes** configuration file (see the Section called *Access with **telnet*** in Chapter 6 for details). Alternatively, **ssh** or **rsh** based access can be used, if these clients are available on the NT/2000/XP host and the Linux/Unix nodes support access with these protocols (see the Section called *Linux/Unix Based Nodes* in Chapter 6).
- If your root computer is a Linux/Unix host, add the string "WindowsNT" to all Windows NT/2000/XP hosts in the **enfuzion.nodes** configuration file

For more details on the root configuration and on the **enfuzion.nodes** file, refer to the Section called *The **enfuzion.nodes** File* in Chapter 6.

Modifying the Installation Defaults

EnFuzion provides default locations for installation directories. These locations can be changed during the installation process. In addition, the default locations can be changed before installation begins. This change of default locations accelerates the installation and removes possible errors, in cases where the default locations need to be changed and EnFuzion is installed on many nodes.

The setup program prompts for the locations of the installation directory and the node working directory during the installation process. Default locations are **C:\enfuzion** and **C:\enfuzion\temp**, respectively. These default directory locations can be changed by modifying the **INSTALL.INI** file in the distribution package, which contains instructions for the **setup** program.

Directory locations are specified in the [Directories] section. The Product line provides the default location for the EnFuzion installation directory. The Temp line provides the default location for the node working directory.

Removal of EnFuzion Software from Windows NT/2000/XP

To remove EnFuzion software, simply execute the program **uninstall**, which is located in the EnFuzion directory. Uninstall removes the EnFuzion Starter Service from the system, deletes EnFuzion files, directories and registry entries.

EnFuzion can also be removed through the Add/Remove Software option in the Control Panel.

Another alternative for removing EnFuzion installations from remote systems is to use the Netsetup program with the **uninstall** command. See the the Section called *Removal of EnFuzion Software from Windows NT/2000/XP* above.)

Windows NT/2000/XP Specific Issues of EnFuzion Operation

This section provides more details about the EnFuzion Starter Service, which is a Windows specific program. It also discusses EnFuzion related performance issues.

Starter Service

The EnFuzion Starter Service runs on each EnFuzion node as a service. It provides remote execution on a Windows NT/2000/XP host. Its primary function is to start the node server, which is the central EnFuzion node component. The Starter Service is automatically installed as part of the standard EnFuzion installation process.

The Starter Service uses the IP port number 17000 to listen for user requests.

The Starter Service produces a log of activities on the system. The log file is called **enfstarter.log** and is located in the EnFuzion temporary directory, which is **C:\enfuzion\temp** by default.

The Starter Service can be configured to refuse connections from hosts that are not trusted. These features are described in the Section called *Trusted Hosts and Executables* in Chapter 7.

The following sections describe the Starter Service configuration file and remote commands.

The service.config File

The Starter Service uses a configuration file, called **service.config**. The file is located in the main EnFuzion directory, which is **C:\enfuzion** by default. The file contains lines with user defined configuration values. Lines that start with "#" are treated as comments.

The following configuration options are provided:

- **loadprofile [true | false]**

  This option determines, whether the user profile is loaded or not, when the EnFuzion node software is started. By default, the user profile is loaded. If the value is **true**, then the service loads the user profile. If available, a roaming user profile is used. If the value is **false**, then the user profile is not loaded. This value might be used in environments that require fast node start, but the loading of user profiles takes a long time.

- **node <user_account> <password> <args>**

  This option starts an EnFuzion node at the computer boot time. The node is started under the <user_account>, using the <password>. <args> are provided as command line arguments to the node. Details are described in the Section called *Starting EnFuzion Nodes at the Computer Boot Time*.

Remote Commands

The Starter Service provides remote management commands. These commands are ASCII strings, terminated by a null character, '\0'. Supported commands are:

- **version**

  Returns the current Starter Service version, terminated by a newline character, '\n', followed by a null character, '\0'.

  Example of a return string:

  ```
  7.2.30\n\0
  ```

- **clearlog**

  Truncates the Starter Service log file in enfstarter.log. It returns the string "OK\n\0" if the log was truncated. Otherwise, it returns:

  ```
  Unable to clear log file "....\enfstarter.log".\n\0
  ```

  Example of a return string:

```
OK\n\0
```

- **getlogs**

  Returns the contents of two node log files. The **enfnodea.log** is printed first, followed by the **enfnodeb.log** file. If the log files do not exist, it returns:

  ```
  Unable to copy file enfnodea.log\n\0
  ```

  See the Section called *Log File Size* in Chapter 7 for more details about the node log files.

The Enfkill Utility

The Enfkill utility provides an emergency termination of EnFuzion nodes. The program causes all EnFuzion nodes to clean up their workspace files and directories and to terminate any EnFuzion activity on nodes.

Enfkill is executed on the root system by:

```
enfkill
```

Enfkill retrieves nodes from the **enfuzion.nodes** file in its working directory. If there is not **enfuzion.nodes** file in the working directory, **enfkill** takes the file from the EnFuzion configuration directory. Default path is **C:\enfuzion\config\enfuzion.nodes**. For each node, it terminates all EnFuzion user tasks and deletes the EnFuzion temporary files.

Use extreme care when executing **enfkill**, since the program terminates all tasks that execute under the EnFuzion user. If a user is interactively logged on the system and the **enfkill** operation is executed on the same machine with the same user name, all user's applications will terminate immediately without letting the user save his work.

Therefore, it is strongly recommended to create a special account to execute EnFuzion nodes on each computer, in order to use the Enfkill program safely.

For security purposes the Enfkill program has been designed not to terminate any program if it is executed on the node under the Administrator account.

Performance Considerations

Although EnFuzion itself requires only limited system resources, user jobs can impose significantly higher demands. The most common causes of poor Windows NT/2000/XP root host performance are insufficient disk space for user input and output files, insufficient RAM when there is an extremely large number of jobs, and a combination of a large number of powerful node systems, short jobs and a slow root host.

The following guidelines can avoid overloading the root host:

- Provide required disk space.
- Provide adequate RAM.
- Provide sufficient processing power on root hosts.
- Provide faster SCSI hard disks, if there is a high volume of disk traffic.

If the Dispatcher starts trashing the disk due to insufficient RAM or processing power, Windows NT/2000/XP might be unable to process networking messages. This trashing sometimes leads to system congestion and application errors. You can choose one of the following options to prevent trashing:

- Terminate some other RAM and CPU consuming applications.
- Reduce the number of EnFuzion nodes.
- Increase the job execution time.
- Increase RAM in your root computer.
- Install a faster processor in your root computer.

Chapter 4. Linux/Unix Installation and Operation

This chapter explains how to install and operate EnFuzion software on Linux/Unix computers. The EnFuzion software consists of the EnFuzion root components, the EnFuzion node components and the EnFuzion submit components. These components must be installed on computers that will act as EnFuzion roots, EnFuzion nodes or EnFuzion submit hosts, respectively. The chapter covers EnFuzion software installation, EnFuzion license installation, enabling Linux/Unix node computers for EnFuzion use, installation of EnFuzion as a service, network installation, installation in a mixed Linux/Unix and Windows NT/2000/XP environment, removal of EnFuzion software and Linux/Unix specific issues of EnFuzion operation.

Installing EnFuzion Software on Linux/Unix

EnFuzion software must be installed on each Linux/Unix computer that will be used as an EnFuzion root, an EnFuzion node or an EnFuzion submit host. The simplest method for installing EnFuzion on Linux/Unix is to execute an installation script from the distribution package. Separate scripts are provided for EnFuzion root installation, for EnFuzion node installation and for EnFuzion submit installation.

The following sections provide details about EnFuzion root, node and submit installation.

Installing EnFuzion Root Software

To install EnFuzion software on a root host, perform the following steps:

- Obtain the EnFuzion distribution package for your Linux/Unix system. Packages are available from the Axceleon (http://www.axceleon.com) Web site.
- Login to the system under the account that will be used to execute EnFuzion root programs. EnFuzion processes on the root system will be executing under this account and the use of a dedicated account for this purpose is encouraged. It is recommended that a new account, called **enfuzion** is created and used to install the EnFuzion software. Since this account name is used by default if EnFuzion is installed as a network service, it simplifies later installation steps.

  Root user privileges are not required at this step of EnFuzion root installation. Root privileges will be required later, if EnFuzion is installed as a service. However, if the installation of the EnFuzion root software is performed by the root user, then installation directories on EnFuzion root will be different to provide a system wide access to EnFuzion binaries.

- Unpack the package to a temporary directory, using the **tar** and **gunzip** utilities on your system. The distribution package and the extraction directories can be deleted after the installation, since they are not required for EnFuzion operation.

- Install EnFuzion root components by executing the **install-root** script in the directory with the extracted EnFuzion distribution files. If the installation is performed by a regular user, the EnFuzion root components are installed to the directory **$HOME/enfuzion**. If the installation is performed by the root user, the components are installed to the directory **/usr/local/enfuzion**. The default installation directory can be changed by providing the target directory as an optional argument to **install-root**.
- Add the path for EnFuzion executables to the PATH environment variable. This step allows you to execute EnFuzion binaries from a command line, without specifying the entire path. If EnFuzion was installed by a regular, non-root user, the default path for executables is **$HOME/enfuzion/bin**. If EnFuzion was installed by a root user, the default executable path is **/usr/local/enfuzion/bin**.
- (optional) EnFuzion can be installed to provide a service on the network. Details are described in the Section called *Installing EnFuzion Root as a Network Service*.

Installing EnFuzion Node Software

To install EnFuzion software on a node host, perform the following steps on each node system:

- Obtain the EnFuzion distribution package for your Linux/Unix system. Packages are available from the Axceleon (http://www.axceleon.com) Web site.
- Login to the system under the account that will be used to execute EnFuzion programs. On EnFuzion node systems, it is recommended that a new account is created and used to install the EnFuzion software. The proposed name and a group for the account are **enfuzion**. EnFuzion processes on the node system will be executing under this account and the use of a dedicated account for this purpose is encouraged.
- Unpack the package to a temporary directory, using the **tar** and **gunzip** utilities on your system. The distribution package and the extraction directories can be deleted after the installation, since they are not required for EnFuzion operation.
- Install EnFuzion node components by executing the **install-node** script in the directory with the extracted EnFuzion distribution files. If several EnFuzion nodes share home directories with NFS or a similar file sharing method, then the EnFuzion node software needs to be installed only on one node.

  If the installation is performed by a regular user, the EnFuzion node components are installed to the directory **$HOME/enfuzion**. If the installation is performed by the root user, the components are installed to the directory **/usr/local/enfuzion**, If the EnFuzion node software is installed under the root account, it is strongly recommended for security reasons that it is not operated under the root account, since user jobs will gain root privileges on the system.

Installing EnFuzion Submit Software

To install EnFuzion software on a submit host, perform the following steps on each system:

- Obtain the EnFuzion distribution package for your Linux/Unix system. Packages are available from the Axceleon (http://www.axceleon.com) Web site.
- Login to the system under the account that will be used to execute EnFuzion programs. EnFuzion software must be installed under all user accounts that will be used to submit jobs. Alternatively, the software can be installed to a common directory that is accessible to all users. There is no need to create any new EnFuzion specific accounts on submit systems.
- Unpack the package to a temporary directory, using the **tar** and **gunzip** utilities on your system. The distribution package and the extraction directories can be deleted after the installation, since they are not required for EnFuzion operation.
- Install EnFuzion submit software by executing the **install-submit** script in the directory with the extracted EnFuzion distribution files. If the installation is performed by a regular user, the EnFuzion submit components are installed to the directory **$HOME/enfuzion**. If the installation is performed by the root user, the components are installed to the directory **/usr/local/enfuzion**. The default installation directory can be changed by providing the target directory as an optional argument to **install-submit**.
- Add the path for EnFuzion executables to the PATH environment variable for each user. This step allows you to execute EnFuzion binaries from a command line, without specifying the entire path. If EnFuzion was installed by a regular, non-root user, the default path for executables is **$HOME/enfuzion/bin**. If EnFuzion was installed by a root user, the default executable path is **/usr/local/enfuzion/bin**.

Reinstalling or Upgrading EnFuzion

If EnFuzion is already installed on the system, you can simply repeat the installation process to upgrade EnFuzion. The installation process will keep the existing configuration files, but it will upgrade all other files. If a previous configuration file already exists, the new file will be copied to the target directory with the **.new** suffix added to its name.

Make sure that you upgrade EnFuzion root, node, and submit software at the same time, using the same EnFuzion release.

Installing EnFuzion on Multiple Computers

If several EnFuzion nodes share home directories with NFS or a similar file sharing method, then the EnFuzion node software can be installed only on one node. It is strongly recommended that EnFuzion node software is not installed or operated under the root user. For an automated installation of EnFuzion across multiple computers, refer to the Section called *Network Installation on Linux/Unix* below.

Handling of Installation Problems

If you experience any problems during installation, you can send e-mail to support@axceleon.com and report the problems. Please include the following information:

- output from the failed installation process
- the contents of the **install-*.log** files, which provide a log of installation events.

Installing EnFuzion License

EnFuzion software will not work without a license file being installed on the root system. EnFuzion node and submit computers do not require a license. To install an EnFuzion license file on the system, rename the file with an EnFuzion license to **enflicense** or **enflicense.txt**, and copy the file to the **config** subdirectory of the EnFuzion installation directory. The default path for the **config** subdirectory is **$HOME/enfuzion/config** for regular users and **/usr/local/enfuzion/config** for the root user.

The **install-root** script can also be used to install an EnFuzion license. If the script finds a file named **enflicense** or **enflicense.txt** in the distribution directory, it installs the license. This capability is useful when installing EnFuzion on a large number of systems, since it automates the license installation step. The license file is simply placed in the unpacked distribution directory before the installation process. The **install-root** program then automatically installs the license while installing other EnFuzion components.

EnFuzion licenses can be purchased from Axceleon. Please contact Axceleon or send an e-mail to sales@axceleon.com for details.

Evaluation EnFuzion licenses are available from the Axceleon (http://www.axceleon.com) Web site.

Enabling Linux/Unix Node Computers for EnFuzion Use

The EnFuzion root software provides powerful capabilities for managing EnFuzion nodes. To fully utilize these capabilities, the EnFuzion root needs to be able to login to the node computer under the user account that was used to install the EnFuzion node software on that system.

EnFuzion supports several methods for remote access to EnFuzion nodes. These methods are industry standard protocols for remote login: **ssh**, **rsh** and **telnet**. In addition, users can provide their own method for use by EnFuzion.

The recommended method for remote access is **ssh**, since it provides the highest level of security and is the easiest to use. The section below provides more details on configuring a Linux/Unix system for use as an EnFuzion node. For configuration of remote access using **rsh** or **telnet**, refer to instructions provided by your Linux/Unix system.

Configuring EnFuzion Nodes for Remote ssh Access

To use **ssh**, EnFuzion requires that the node allows a login from the root system without requesting a password. This method enhances security over the **telnet** based login, since no clear text passwords are sent over the network and the root is authenticated and authorized.

The **ssh** protocol uses a public key method. The root generates a public and a private key. The private key is kept secret on the root, while the public key is stored on the node and is used at the login time to authenticate the root.

The procedure to generate the keys on the root and store the public key on the node is described below. It must be performed for each EnFuzion node system that is running a Linux/Unix operating system.

- On the node, the **sshd** daemon must be running
- On the root, generate keys. This step is done only once. If the keys have been already generated, you can skip the step below.

```
# generate keys, store public key to ~/.ssh/id_dsa.pub
# use empty passphrase
ssh-keygen -d -b 1024 -C <local_user>@<root_host.root_domain>
```

- On the root, copy the public key to the node system.

```
scp ~/.ssh/id_dsa.pub <node_user>@<node_host.node_domain>:
```

- On the node, add the public key to the list of authorized keys. Authorized keys are usually stored in file **~/.ssh/authorized_keys**. On some systems, **~/.ssh/authorized_keys2** must be used instead.

```
mkdir .ssh
chmod 0700 .ssh
cat id_dsa.pub >> ~/.ssh/authorized_keys
chmod 0644 ~/.ssh/authorized_keys
```

- On the root, test the configuration, **ssh** should login immediately without requesting a password:

```
ssh <node_user>@<node_host>.<node_domain>
```

If all the steps above are completed successfully, then the node is ready to be used by EnFuzion.

Installing EnFuzion Root as a Network Service

The Dispatcher can be installed as a network service, which means that it automatically started at the computer boot time and available to remote users over the network. This configuration is suitable for environments where one Dispatcher is used by multiple users, and jobs are submitted remotely from the user computers.

EnFuzion provides a script for a straightforward network service installation on Linux and Mac OS X operating systems. The installation must be performed manually on other operating systems. The installation steps on Linux and Mac OS X operating systems and the manual installation on other systems are described below.

Network Service Installation on Linux and Mac OS X

To install EnFuzion as a network service on the root system with Linux or Mac OS X, perform the following steps:

- Install EnFuzion root software on the system as described in the Section called *Installing EnFuzion Root Software*.
- Login to the system under the root account.
- Install EnFuzion service by executing the **install-service** script in the directory with the extracted EnFuzion distribution files:

  ```
  ./install-service
  ```

  The script assumes that the EnFuzion root software is installed under the **enfuzion** user and in the default installation directory. Otherwise, on Linux only, the user, the group, and the installation directory can be specified by executing:

  ```
  ./install-service  <user>  <group>  [ <directory> ]
  ```

  For security reasons, it is recommended that the use of the root account is avoided and that the EnFuzion root software is not running under the root account.

  On Linux, the **install-service** script installs the EnFuzion init script, which is called **enfuzion**, enables the script execution at the boot time, creates the EnFuzion working directory **/var/local/enfuzion**, configures the EnFuzion root to accept node connections, and sets the service port to 10102. The default directory can be changed by editing the value of **DISPATCHER_WORK_DIR** in the **/etc/init.d/enfuzion** script. The default port can be changed by editing the value of **DISPATCHER_PORT** in the **/etc/init.d/enfuzion** script.

  On Mac OS X, the **install-service** script installs EnFuzion startup scripts in the directory **/Library/StartupItems/EnFuzion**, enables the script execution at the boot time, creates the EnFuzion working directory **/Users/enfuzion/enfuzion/work**, configures the EnFuzion root to accept node connections, and sets the service port to 10102. The default values can be changed by editing EnFuzion startup scripts in directory **/Library/StartupItems/EnFuzion**.

Manual Network Service Installation

To install EnFuzion manually as a network service on the root system, perform the following steps:

- Install EnFuzion root software on the system as described in the Section called *Installing EnFuzion Root Software*.
- Login to the system under the account that was used for installation.
- Create the Dispatcher working directory under the account that was used to install EnFuzion root software.
- Start the Dispatcher in the working directory with command:

```
enfdispatcher -m -r -d -p 10102
```

This command starts the service on port 10102. Change the number to provide the EnFuzion service on a different port.

- (optional)To start the Dispatcher at the boot time, include its execution in the system init sequence. Details depend on your system. Although the system init sequence is usually performed by the root user account, it is highly discouraged to run EnFuzion service under the root user. To avoid this problem, set EnFuzion executables to run under a non-privileged user account. The recommended options for the Dispatcher are:

  ```
  enfdispatcher -m -r -d -p 10102
  ```

  This command starts the service on port 10102. Change the number to provide the EnFuzion service on a different port.

  On the Linux, the init script is called **enfuzion-init-script** and is located in the **config** directory of the Linux distribution package. This script can be used as a template for other platforms. Although there is no guarantee, the Linux service installation and the init script might work on other common Linux distributions.

Starting EnFuzion Nodes at the Computer Boot Time

EnFuzion nodes can be configured as daemons that start automatically at the computer boot time. These nodes can then connect to an EnFuzion root or wait to be contacted, depending on the root and node configuration. This automated node start is especially suitable for highly flexible environments, where node or even root computers change often. It simplifies EnFuzion configuration by eliminating the need for the **enfuzion.nodes** file.

EnFuzion provides a script for a straightforward node installation as a daemon on Linux and Mac OS X operating systems. The installation must be performed manually on other operating systems. The installation steps on the Linux and Mac OS X operating systems and the manual installation on other systems are described below.

Nodes that are started by the EnFuzion script are configured to connect to an EnFuzion root instead of waiting for a connection from the root. The EnFuzion root must be enabled for this feature to work. By default, connections from external nodes are rejected by the EnFuzion root and nodes will fail to connect. The **rootport** root option, which is described in the Section called *Port Number for Node Connections* in Chapter 6, enables the EnFuzion root for external node connections. Make sure that the **rootport** option is configured on the EnFuzion root, before starting the nodes as described here. For additional details on connecting a node to the root, see the Section called *Nodes with No Root Control, Connection Initiated by the Node* in Chapter 6. For configuration details on connecting the root to a node, see the Section called *Nodes with No Root Control, Connection Initiated by the Root* in Chapter 6.

Installing EnFuzion Node as a Daemon on Linux and Mac OS X

Perform the following steps to install an EnFuzion node, so that it is started automatically as a daemon on Linux or Mac OS X at the computer boot time:

- Install EnFuzion node software on the system as described in the Section called *Installing EnFuzion Node Software*.
- Login to the system under the root account.
- Install the EnFuzion node daemon by executing the **install-svcnode** script in the directory with the extracted EnFuzion distribution files:

  ```
  ./install-svcnode <root_host>
  ```

  Replace <root_host> with the network address of the EnFuzion root host.

  The script assumes that the EnFuzion node software is installed under the **enfuzion** user and in the default installation directory. Otherwise, on Linux only, the user, the group, and the installation directory can be specified by executing:

  ```
  ./install-svcnode  <root_host> <user>  <group>  [ <directory> ]
  ```

  For security reasons, it is recommended that the use of the root account is avoided and that the EnFuzion node software is not running under the root account.

  On Linux, the **install-svcnode** script installs the EnFuzion init script, which is called **enfnode**, and enables the script execution at the boot time.

  On Mac OS X, the **install-svcnode** script installs EnFuzion startup scripts in the directory **/Library/StartupItems/EnFuzionNode**, and enables the script execution at the boot time.

Manual Node Daemon Installation

To install EnFuzion node manually as a daemon, perform the following steps:

- Install EnFuzion node software on the system as described in the Section called *Installing EnFuzion Node Software*.
- Login to the system under the account that was used for installation.
- Change the current working directory to the directory with the EnFuzion node installation. This directory is **~/enfuzion** by default.
- Start the nodeserver in the working directory with command:

  ```
  enfnodeserver -b -d -n 0 0
  ```

  The node server connects to an EnFuzion root on the local network. Change "0 0" to "<host> <port>" to connect the node server to an EnFuzion root on a specific host and port.

- (optional)To start the node server at the boot time, include its execution in the system init sequence. Details depend on your system. Although the system init sequence is usually performed by the root user account, it is highly discouraged to run EnFuzion node daemon under the root user. To avoid this problem, set EnFuzion executables to run under a non-privileged user account. The recommended options for the node server are:

  ```
  enfnodeserver -b -d -n 0 0
  ```

  The node server connects to an EnFuzion root on the local network. Change "0 0" to "<host> <port>" to connect the node server to an EnFuzion root on a specific host and port.

  On Linux, the init script is called **enfuzion-init-node** and is located in the **config** directory of the Linux distribution package. This script can be used as a template for other platforms. Although there is no guarantee, the Linux daemon installation and the init script might work on other common Linux distributions.

Network Installation on Linux/Unix

The EnFuzion distribution package provides the program Enfinstall, which is able to install EnFuzion on remote Linux/Unix hosts from a central location.

Enfinstall Program

The Enfinstall program can be used to install EnFuzion on remote systems, without any need to access the system's keyboard or monitor. The program can also be used to verify an EnFuzion configuration and copy the options file to the nodes.

The Enfinstall program is called with a command option:

```
enfinstall <command>
```

Enfinstall Commands

- **enfuzion**

  Installs EnFuzion node software on node systems. /usr/local/enfuzion.

- **verify**

  Accesses nodes and verifies their installation.

- **options**

  Copies the **enfuzion.options** file to node systems.

- **collect**

  Collects the information about EnFuzion nodes.

Remote Installation

Enfinstall installs EnFuzion nodes in a heterogeneous network. The program executes on your local root host and installs EnFuzion nodes over the network. The program Enfinstall automatically detects the type of each remote host and installs the corresponding executables.

The program Enfinstall works only on supported Linux/Unix platforms. See the Section called *Installation in a Mixed Linux/Unix and Windows NT/2000/XP Environment* below to install EnFuzion in mixed environments.

Follow these steps to perform the installation:

- Obtain the EnFuzion distribution packages for all Linux/Unix systems in your EnFuzion configuration. Packages are available from the Axceleon (http://www.axceleon.com) Web site.
- Copy all the packages to the same directory on the root system. Unpack the packages in that same directory, using the **tar** and **gunzip** utilities on your system. The distribution packages and the extraction directories are not required for EnFuzion operation, and you can delete them after the installation.
- Go to the extraction directory that contains executables of your root system.
- Install EnFuzion root software on the local system by executing the **install-root** script in the current directory.
- Add the path for EnFuzion executables to the PATH environment variable. This step allows you to execute Enfinstall without specifying the entire path. The default path for EnFuzion executables is **$HOME/enfuzion/bin**, if EnFuzion was installed by a non-root user and **/usr/local/enfuzion/bin** otherwise.
- Prepare the configuration file install.nodes, which contains a list of node hosts.

  The configuration file **install.nodes** contains a description of your EnFuzion network configuration. Each line describes one node. It contains a remote host, a user account on that host, a password and an optional remote access protocol. For example, to install EnFuzion on three hosts called "host1", "host2", and "host3" under user account "user1", the **install.nodes** file looks as follows:

```
host1   user1   password   [ remote_access ]
host2   user1   password   [ remote_access ]
host3   user1   password   [ remote_access ]
```

Optional remote access protocol can be one of "Unix", "UnixRsh", or "ssh". The **install.nodes** file has the same syntax as the **enfuzion.nodes** file, which is used after the installation to connect the EnFuzion root system with EnFuzion nodes. More details about the **enfuzion.nodes** file can be found in the Section called *The **enfuzion.nodes** File* in Chapter 6.

- Install EnFuzion node executables on remote hosts with the command:

  ```
  enfinstall enfuzion
  ```

  EnFuzion is installed in the directory **~/enfuzion** on the node for all accounts, except the root account. If the the root user is specified in the **install.nodes** file, EnFuzion is installed in the directory **/usr/local/enfuzion** on the node host.

  The Enfinstall program must be run from the distribution directory with executables for your local host. Package directories for all nodes in your EnFuzion configuration must be in the same parent directory.

  By default, the Enfinstall program uses the standard **ftp** protocol to copy files. Some **ftp** servers (e.g., Sun Solaris) do not support the command to set execution permissions. In such cases, Enfinstall issues a warning. If you get such a warning, make sure that the execution permissions for EnFuzion node executables are set on the node hosts. The files that require attention will be reported by Enfinstall on the screen.

Testing Remote EnFuzion Operation

Verify the installation of EnFuzion nodes with the command:

```
enfinstall verify
```

This command verifies that EnFuzion is correctly installed. For all EnFuzion nodes in your configuration, it starts the node, establishes a connection and reports whether the node is operational. Alternatively, it reports any errors encountered.

If you decide to use the load monitoring features of EnFuzion, install the **enfuzion.options** file with the command:

```
enfinstall options
```

This command will copy the options file to remote hosts. It will ask you for the installation directory. The use of the default install directory is recommended. The file **enfuzion.options** must be in the current directory. Details on the contents of the **enfuzion.options** file can be found in the Section called *The **enfuzion.options** File* in Chapter 7.

Installation in a Mixed Linux/Unix and Windows NT/2000/XP Environment

The Enfinstall program works only on supported Linux/Unix platforms. It does not support EnFuzion installation on Windows NT/2000/XP hosts. A separate installation program is provided for Windows NT/2000/XP hosts. See Chapter 3 for more information about installing EnFuzion on Windows.

Perform the following steps to install EnFuzion in a mixed Linux/Unix and Windows NT/2000/XP environment:

- Install EnFuzion on all Linux and Unix hosts.
- Install EnFuzion on all Windows NT/2000/XP hosts.
- Include all hosts in your configuration file **enfuzion.nodes**.
- If your root computer is a Unix host, add the string "WindowsNT" to all Windows NT/2000/XP hosts in the configuration file.
- If your root computer is a Windows NT/2000/XP host, add the string "Unix" to all Unix hosts in the configuration file. In this case, the **telnet** remote access must be enabled on all Linux/Unix nodes.

Removal of EnFuzion Software from Linux/Unix

To remove EnFuzion software, simply delete the EnFuzion installation directory. The installation directory is **$HOME/enfuzion** for regular accounts and **/usr/local/enfuzion** for the root account.

Linux/Unix Specific Issues of EnFuzion Operation

This section provides more details about EnFuzion performance considerations on Linux/Unix systems.

Performance Considerations

EnFuzion configurations with a very large number of EnFuzion nodes can require more resources on the root host than is normally configured by standard system configurations, especially on older Linux/Unix systems. In these cases, it might be necessary to configure the root Linux/Unix system with a larger resource limit.

Two limitations are often encountered, the total number of processes on the host and the number of opened file descriptors per process.

On the root, EnFuzion executes three processes to handle root tasks. In addition, there may be at most one process per each job executing. Job processes on the root are only created when required by the job. The total number of processes can thus be as large as n+3, where n is the number of EnFuzion nodes. If the process table is not large enough to accommodate all processes, some jobs may fail to complete

successfully. Make sure that the process table on the root system is large enough to accommodate your job workload requirements.

The Dispatcher requires a small number (usually less than 10) of task descriptors to handle file I/O and one file descriptor for each EnFuzion node. The maximum number of opened file descriptors is thus n+10, where n is the number of EnFuzion nodes. If a process on the root host is not allowed to have that many opened descriptors, some nodes will fail to execute jobs. Make sure that the process limit for opened file descriptors is large enough to accommodate your job workload requirements.

Chapter 5. Submit Configuration

This chapter provides details about the EnFuzion submit host configuration.

There are only minimal configuration requirements for the submit host. If only a standard web browser is being used to communicate with the EnFuzion service, then there are no configuration requirements on the submit host.

Otherwise, the submit host must be configured with the EnFuzion service address. By default, the EnFuzion service address is **localhost:10102**. If the service address is different from the default, it must be specified in the **submit.config** file.

Details about the **submit.config** file are described in the rest of the chapter.

Specifying the EnFuzion Service Address

When EnFuzion is provided as a service over the network, the submit host must know its address. By default, the service address of **localhost:10102** is used. Otherwise, the address is specified in the **submit.config** file.

The submit.config File

The **submit.config** can be placed either in the local directory or in the **config** EnFuzion directory.

The file contains the EnFuzion network address in the following format:

```
<host_name>:<port_number>
```

<host_name> specifies the EnFuzion root host IP address and <port_number> specifies the Dispatcher API port number, which is used by the submit programs.

An example file is:

```
enfuzion.domain.com:10102
```

Chapter 6. Root Configuration

This chapter provides details about the EnFuzion root configuration.

The most important aspect of the EnFuzion root configuration is how the root establishes the communication with EnFuzion nodes. This communication method is dependent on the node type. EnFuzion implements a wide range of different node types, which allows EnFuzion to be optimally configured for environments with varying requirements. Some nodes are configured in the **enfuzion.nodes** file. This file is often the only configuration file that is required to run EnFuzion.

The EnFuzion root also contains several configuration options, which can be tuned for specific user environments in order to improve performance or security aspects of EnFuzion operation. These options are provided in the **root.options** file. This file is optional for running EnFuzion and can contain only options that are relevant for a particular EnFuzion installation.

Several configuration files describe how EnFuzion deals with users. These files are: **users**, which specifies how user identities are assigned from user identification strings that are provided by the submit computers; **groups**, which specifies groups membership for users; **admins**, which contains a list of users that have EnFuzion administrator privileges; and **user.accounts**, which contains rules for user account handling on EnFuzion nodes.

The EnFuzion root software provides additional security features that enhance system provided security. These features include the capability to remove clear text passwords from **enfuzion.nodes** and a private-public key method to authenticate the root to the nodes.

The rest of the chapter describes details about node types, configuring **root.options**, configuring user related options, and root based security features.

Specifying EnFuzion Node Type

EnFuzion implements several node types. These types can be classified depending on:

- the control of the EnFuzion node server. Node server, which is the main EnFuzion process on nodes, can be either controlled or not controlled by the EnFuzion root;
- the connection between the EnFuzion root and EnFuzion node processes. The connection can be initiated from the root computer or from a node computer.

Each node can be configured independently of other nodes.

The following table summarizes node types according to the root control and the connection.

Table 6-1. Node Types

	Root Control	No Root Control

	Root Control	No Root Control
Root Connection	Local Windows Linux/Unix Custom Start	Direct
Node Connection	WindowsNode	Dynamic Static

The simplest node type to configure and use is the local node. These nodes execute on the EnFuzion root computer and do not require any remote execution. They are useful for learning about EnFuzion, for application testing and for job scheduling on the local computer.

Windows and Linux/Unix based nodes are the most commonly used node types. These nodes provide a single point of control for distributed execution. Custom start nodes are used only in specialized applications.

Direct node type is useful for networks, where the connection between the root and the node goes through a firewall. It is not commonly used.

The WindowsNode type is controlled by the root, but the connection is initiated from the node. From the EnFuzion point of view, this node type behaves the same as the Windows type. Its advantage is that it provides better compatibility with firewalls and anti-virus programs on Windows nodes. If an anti-virus program is active on the node, then the WindowsNode type is recommended instead of the Windows type.

Dynamic node type is the easiest to manage among remote node types, since the root and nodes configure themselves. This type is especially suitable for compute clusters. Dynamic nodes must be on the same network as the EnFuzion root. If nodes are on a different network than the root, then similar benefits can be obtained by the static node type. An additional benefit of dynamic and static nodes is that they can be configured to operate autonomously, as described in the Section called *Autonomous Node Operation* and the Section called *Bind* in Chapter 7.

Sections below describe the **enfuzion.nodes** file, which is used on the root to describe most of the node types, and then give details about each of the node types.

The enfuzion.nodes File

Most node types are specified in the **enfuzion.nodes** file. The file defines how the Dispatcher on the EnFuzion root connects with its nodes. Alternatively, node descriptions can be dynamically added and

removed through the API, described in detail in the Section called *Application Programming Interface* in Chapter 10.

EnFuzion checks for the **enfuzion.nodes** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

For each node, **enfuzion.nodes** contains a line, describing the host name, the user name and the method used to establish a connection between the root and the node. A different method can be used by each node to connect to the root, which makes it easy to combine hosts with different operating systems and configurations. Lines that start with "#" are treated as comments.

Nodes with Root Control, Connection Initiated by the Root

Nodes in this category are controlled by the EnFuzion root. The EnFuzion root starts or terminates the node server process. The node server is terminated, if the EnFuzion root is terminated. The connection between the root and a node is initiated by the root. These nodes must be described in the **enfuzion.nodes**. No configuration options are required on the nodes.

Nodes can be either local nodes (see the Section called *Local Nodes*), Windows nodes (see the Section called *Windows Based Nodes*), Linux/Unix nodes (see the Section called *Linux/Unix Based Nodes*), or custom nodes (see the Section called *Custom Node Start*). Local nodes execute on the same host and under the same user as the EnFuzion root. Windows nodes execute on Windows computers and EnFuzion provides a standard way through the EnFuzion Starter Service to start them. Linux/Unix nodes execute on Linux/Unix computers and EnFuzion uses one of the standard protocols - ssh, rsh, or telnet, to start a node. For environments with special requirements, EnFuzion supports a custom node start through a user provided script or a program. Each individual node type is described in detail in the section below.

Local Nodes

Local nodes are executed on the same computer and under the same user account as the EnFuzion root. Local nodes are specified in the **enfuzion.nodes** file with a line:

```
localhost
```

Local nodes are useful for testing purposes or when EnFuzion is used for job queue scheduling on a single computer.

Windows Based Nodes

In general, EnFuzion nodes are executing on a computer that is different than the EnFuzion root computer and possibly under a different user account. By default, EnFuzion uses the EnFuzion Starter Service on Windows NT/2000/XP, which significantly simplifies configuration of Windows nodes.

For each Windows based node, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   <password>
```

The items, <host_name>, <user_name>, and <password> specify the host name, the user name under which EnFuzion executes programs and the user password on that host.

<user_name> can contain an optional <domain>. It takes one of the following forms:

```
<user_account>
<domain>\<user_account>
<user_account>@<domain>
```

If only <user_account> is provided, the node host name is used for the domain name.

If the root host is a non-Windows computer, but the node host is a Windows computer, then the line format is as follows:

```
<host_name>   <user_name>   <password>  WindowsNT
```

The example below details an **enfuzion.nodes** file that specifies EnFuzion nodes on four computers called ballet, swanlake, mandarin, and firebird. EnFuzion uses "enfuzion" as its user to execute programs with the password "enftest". All nodes, including the root, are Windows-based hosts.

Example of a Windows root and Windows nodes:

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest
swanlake.domain.com     enfuzion   enftest
mandarin.domain.com     enfuzion   enftest
firebird.domain.com     enfuzion   enftest
```

If the root is a non-Windows host, but the nodes are Windows-based, then the example above would look like the following.

Example of a non-Windows root and Windows nodes:

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest   WindowsNT
swanlake.domain.com     enfuzion   enftest   WindowsNT
mandarin.domain.com     enfuzion   enftest   WindowsNT
firebird.domain.com     enfuzion   enftest   WindowsNT
```

Some Windows installations might require that a domain name be specified for a node, in addition to the host name and the user name. In that case, the domain name can be specified along with the user name. The following is an example of the corresponding syntax for the **enfuzion.nodes** file:

```
<host_name> [<domain_name>\]<user_name> [ <password> ] [WindowsNT]
```

A domain name is optional, and if not specified, the local domain is used.

Passwords are required by EnFuzion to start the execution of nodes on Windows hosts. If the same password is shared among several computers, its handling in **enfuzion.nodes** can be simplified. When a host has a password that is already specified for a previous host in the configuration file, its password can be written as a "$" symbol, followed by the previous host's name, indicating that both hosts use the same

password. In clusters with a large number of nodes, this feature can significantly simplify password changes.

The original example above, using shared passwords, would look like the following.

Example of shared passwords:

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest
swanlake.domain.com     enfuzion   $ballet.domain.com
mandarin.domain.com     enfuzion   $ballet.domain.com
firebird.domain.com     enfuzion   $ballet.domain.com
```

In some environments, it may not be desirable to keep clear text passwords in the **enfuzion.nodes** file. EnFuzion supports the use of encrypted passwords, which is described in the Section called *Encrypted Passwords in enfuzion.nodes*.

Linux/Unix Based Nodes

EnFuzion provides several methods to connect a root to a Linux/Unix-based host. These methods include **ssh**, **rsh** or **telnet**.

*Access with **ssh***

The **ssh** method is recommended, since it provides the highest level of security and is the easiest for installation. If the **ssh** method is used to connect to a remote node, then the root and the node must be configured for RSA based authentication. RSA based authentication allows the root to access a node without sending its password in a clear text format. The details of configuring the root and nodes for the **ssh** access are described in the Section called *Configuring EnFuzion Nodes for Remote **ssh** Access* in Chapter 4.

The example below assumes that the root and nodes have been configured to use RSA based authentication. If the root and nodes are not configured for RSA based authentication, then the Dispatcher and other programs prompt for a password. This method is not recommended, since it precludes batch execution of the Dispatcher.

For each Linux/Unix based node with **ssh**, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   dummy   ssh
```

<host_name>, and <user_name> specify the host name, and the user name under which EnFuzion executes programs. Since a password is not required, the third field, which normally contains a password, is ignored.

The example below shows an enfuzion.nodes file that specifies EnFuzion nodes on four computers called ballet, swanlake, mandarin, and firebird. EnFuzion uses "enfuzion" as its user to execute programs. **ssh** is used to connect to all nodes and is configured for RSA based authentication for accesses from the root.

*Example of Linux/Unix-based nodes with **ssh**, using RSA:*

```
# this file describes my cluster
ballet.domain.com       enfuzion   dummy   ssh
```

```
swanlake.domain.com     enfuzion   dummy   ssh
mandarin.domain.com     enfuzion   dummy   ssh
firebird.domain.com     enfuzion   dummy   ssh
```

*Access with **rsh***

rsh can used to connect to nodes instead of **ssh** by replacing the **ssh** keyword with the Unixrsh keyword. Nodes must be configured so that no password is required for **rsh** access to nodes.

For each Linux/Unix based node with **rsh**, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   dummy   Unixrsh
```

If **rsh** access is used instead of **ssh**, the example above would look like the following.

*Example of Linux/Unix-based nodes with **rsh**:*

```
# this file describes my cluster
ballet.domain.com       enfuzion   dummy   Unixrsh
swanlake.domain.com     enfuzion   dummy   Unixrsh
mandarin.domain.com     enfuzion   dummy   Unixrsh
firebird.domain.com     enfuzion   dummy   Unixrsh
```

*Access with **telnet***

Telnet, a standard Linux/Unix protocol, is another method that EnFuzion can use to connect to nodes. This method is the least recommended, since it provides the lowest level of security, because clear text password are sent over the network.

For each **telnet** based node, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   <password>
```

<host_name>, <user_name>, and <password> specify the host name, the user name under which EnFuzion executes programs and the user password on that host.

If the root host is a non-Linux/Unix computer, but the node host is a **telnet** based node, then the line format is as follows:

```
<host_name>   <user_name>   <password>  Unix
```

The example below shows an **enfuzion.nodes** file that specifies EnFuzion nodes on four computers called ballet, swanlake, mandarin, and firebird. EnFuzion uses "enfuzion" as its user to execute programs with the password "enftest". All nodes are telnet-based hosts, and the root is a Linux/Unix-based host.

*Example of a Linux/Unix root and **telnet** nodes:*

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest
swanlake.domain.com     enfuzion   enftest
mandarin.domain.com     enfuzion   enftest
firebird.domain.com     enfuzion   enftest
```

If the root is a non-Linux/Unix host, but the nodes are telnet-based, then the example above would look like the following.

*Example of a non-Linux/Unix root and **telnet** nodes:*

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest   Unix
swanlake.domain.com     enfuzion   enftest   Unix
mandarin.domain.com     enfuzion   enftest   Unix
firebird.domain.com     enfuzion   enftest   Unix
```

Passwords are required by EnFuzion to start the execution of nodes on telnet- based hosts. If the same password is shared among several computers, its handling in **enfuzion.nodes** can be simplified. When a host has a password that is already specified for a previous host in the configuration file, its password can be written as a "$" symbol, followed by the previous host's name, indicating that both hosts use the same password. In clusters with a large number of nodes, this feature can significantly simplify password changes.

The original example above, using shared passwords, would look like the following.

Example of shared passwords:

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest
swanlake.domain.com     enfuzion   $ballet.domain.com
mandarin.domain.com     enfuzion   $ballet.domain.com
firebird.domain.com     enfuzion   $ballet.domain.com
```

In some environments, it may not be desirable to have clear text passwords in the **enfuzion.nodes** file. EnFuzion supports the use of encrypted passwords, as described elsewhere in the Section called *Encrypted Passwords in enfuzion.nodes*.

Custom Node Start

Although EnFuzion provides a wide variety of methods to connect to a node, some environments require custom methods. EnFuzion supports custom remote execution, which allows users to start remote EnFuzion nodes through a user provided program, instead of using any of the standard methods.

For each node with a custom method, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   <password>   command   <start_command>
```

The <host_name>, <user_name>, and <password> specify the host name, the user name under which EnFuzion executes programs and the user password on that host. <start_command> is the command used to start the node.

The command is called on the root host whenever the node is started by EnFuzion. It is provided with the following options:

```
<host_name> <user_name> <password> nodestart <node_command>
```

The first three arguments, <host_name>, <user_name>, and <password>, are the same as in **enfuzion.nodes** file. The <node_command> contains the command to be executed by EnFuzion on the node system to start the EnFuzion node software.

The <node_command> starts the **enfnodeserver** executable on the node (see the Section called *Enfnodeserver* in Chapter 11 for more details about the program). The path to the program is different depending on whether <user_name> is the root account or a regular user account. If <user_name> is the root account, <node_command> contains the following string:

```
cd /usr/local/enfuzion ; ./enfnodeserver -d -p <root_IP> <root_port>
```

The <root_IP> is the IP address of the root computer and <root_port> is the port number to which the node connects to exchange files or execute commands on the root system.

For all non-root accounts, <node_command> contains the following string:

```
cd ; cd enfuzion ; ./enfnodeserver -d -p <root_IP> <root_port>
```

The user command, specified in <start_command> can use the command line above and its own method to start the node. The node program, called Enfnodeserver, must be started with the same options as the standard EnFuzion command.

With the **-d** option, **enfnodeserver** creates a child, which runs as a daemon. It is important that <start_command> waits for the parent **enfnodeserver** process to exit. Otherwise, the parent can be terminated before the child daemon is started successfully, causing the node start procedure to fail. One simple solution is to wait for the prompt after the node start command is issued.

Specifying Node Port Number

By default, the node port number used by the root for connection is assigned dynamically. If the network traffic between the root and a node is controlled by a firewall, then dynamic port assignment might not be compatible with the firewall. In such cases, the node can be configured in **enfuzion.nodes** with a static port assignment.

A static port is specified for a node by adding the port option to its line in **enfuzion.nodes**:

```
<host_name>   <user_name>      ...   port   <port_number>
```

The port option works with any method for starting a node. An example of a static port assignment for nodes started with **ssh** is shown below.

Example:

```
# nodes are started with ssh and use port 1234
ballet.domain.com      enfuzion   dummy   ssh   port   1234
swanlake.domain.com    enfuzion   dummy   ssh   port   1234
mandarin.domain.com    enfuzion   dummy   ssh   port   1234
firebird.domain.com    enfuzion   dummy   ssh   port   1234
```

Nodes with No Root Control, Connection Initiated by the Root

Nodes in this category are not controlled by the root. The node server process is started independently of the EnFuzion root. The EnFuzion root does not control the start or termination of these nodes, although it

can terminate the node connection. If the EnFuzion root is terminated, the node server termination depends on the node configuration. Since the connection between the root and a node is initiated by the root, these nodes must be described in the **enfuzion.nodes**. Node configuration options must be set up properly for these nodes.

The following section provides details about the only node type in this category - direct nodes.

Direct Nodes

For all node types described so far, the root host is responsible for starting a node. Direct nodes must be started independently, either as a daemon on the node host, manually or using some other user defined method.

For each direct node, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   dummy   dummy   nostart   port   <port_number>
```

The <host_name> and <port_number> specify the host name and the port number to use for connection to that host.

The example below shows an **enfuzion.nodes** file that specifies EnFuzion nodes on four computers called ballet, swanlake, mandarin, and firebird. Nodes are available on port 1234.

Example:

```
# direct nodes, using port 1234
ballet.domain.com      dummy   dummy   nostart   port   1234
swanlake.domain.com    dummy   dummy   nostart   port   1234
mandarin.domain.com    dummy   dummy   nostart   port   1234
firebird.domain.com    dummy   dummy   nostart   port   1234
```

It is the user's responsibility to start the nodes and make them available at the specified hosts and ports. For a direct node, the node server must be started with the following command line:

```
enfnodeserver -c <port> -r -h
```

The <port> is the port number on which the node server waits for a connection, which would be 1234 in the example above. The same configuration can be achieved with the following configuration options in the **node.config** file on the node:

```
nodeport <port>
report off
hello off
```

In the example above, the node server terminates after the first connected root terminates. If the node server needs to wait for another connection from the EnFuzion root, it must be started with the following command line:

```
enfnodeserver -c <port> -r -h -b
```

With the **-b** option, the node server is started in the batch mode. It never exits and is always ready for a connection from the root. The same configuration can be achieved with the following configuration options in the **node.config** file on the node:

```
nodeport <port>
report off
hello off
batch on
```

Options in the **node.config** file are described in more detail in the Section called *Specifying Node Configuration Options* in Chapter 7.

EnFuzion provides a simple installation of the node server, so that it is started automatically at the computer boot time. For Windows, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 3. For Linux/Unix, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 4.

Nodes with Root Control, Connection Initiated by the Node

Nodes in this category are controlled by the EnFuzion root. The EnFuzion root starts or terminates the node server process. The node server is terminated, if the EnFuzion root is terminated. Although nodes are controlled by the EnFuzion root, the connection between the root and a node is initiated by the node, not by the root. These nodes must be described in the **enfuzion.nodes**. No configuration options are required on the nodes.

This option is currently supported only for nodes, running Windows. No equivalent option is provided for Unix based nodes.

WindowsNode Type

For each WindowsNode type node, the **enfuzion.nodes** file contains a line in the following format:

```
<host_name>   <user_name>   <password>   WindowsNode
```

The items, <host_name>, <user_name>, and <password> specify the host name, the user name under which EnFuzion executes programs and the user password on that host.

<user_name> can contain an optional <domain>. It takes one of the following forms:

```
<user_account>
<domain>\<user_account>
<user_account>@<domain>
```

If only <user_account> is provided, the node host name is used for the domain name.

The example below details an **enfuzion.nodes** file that specifies EnFuzion nodes on four computers called ballet, swanlake, mandarin, and firebird. EnFuzion uses "enfuzion" as its user to execute programs with the password "enftest".

Example of a WindowsNode type nodes:

```
# this file describes my cluster
ballet.domain.com       enfuzion   enftest   WindowsNode
swanlake.domain.com     enfuzion   enftest   WindowsNode
mandarin.domain.com     enfuzion   enftest   WindowsNode
firebird.domain.com     enfuzion   enftest   WindowsNode
```

In some environments, it may not be desirable to keep clear text passwords in the **enfuzion.nodes** file. EnFuzion supports the use of encrypted passwords, which is described in the Section called *Encrypted Passwords in enfuzion.nodes*.

Nodes with No Root Control, Connection Initiated by the Node

Nodes in this category are not controlled by the root. The node server process is started independently of the EnFuzion root. The EnFuzion root does not control the start or termination of these nodes, although it can terminate the node connection. If the EnFuzion root is terminated, the node server termination depends on the node configuration. These nodes are not described in the **enfuzion.nodes**, since the connection between the root and a node is initiated by the node. Node configuration options must be set up properly for these nodes.

This node type can be useful for compute clusters, when EnFuzion is integrated with batch schedulers, when the networking environment is limited by firewalls or when the nodes change often.

Dynamic Nodes

Dynamic nodes automatically find an EnFuzion root on the same network. The EnFuzion root periodically broadcasts its address, which can be obtained by a new node. There is not need to configure any network addresses on the root or on the node.

The root and the node must both be configured for dynamic nodes. On the root, the **rootport** option must be specified in the **root.options** file to provide a root port number for node connections (see the Section called *Port Number for Node Connections*). By default, the port is not open.

Dynamic nodes are started with the following command line:

```
enfnodeserver -b -d -n 0 0
```

This command line starts the node server as a background daemon in batch mode. The node server waits for an address broadcast from an EnFuzion root and then connects to the root. If the connection with the root is terminated, the node server waits for the next root address.

The **-b** and **-n 0 0** options can be also specified in the **node.config** file on the node instead of on the command line:

```
connect on
batch on
```

Optionally, the **connectretry** and **connectdelay** options can be specified to change the EnFuzion provided default values.

EnFuzion provides a simple installation of the node server, so that it is started automatically at the computer boot time. For Windows, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 3. For Linux/Unix, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 4.

Options in the **node.config** file are described in more detail in the Section called *Specifying Node Configuration Options* in Chapter 7.

Static Nodes

Static nodes are similar to dynamic nodes, except that the EnFuzion root network address must be configured. Static nodes are used if the root and nodes are on different networks or if the broadcast of the root network address is not desirable.

The root and the node must both be configured for static nodes. On the root, the **rootport** option must be specified in the **root.options** file to provide a root port number for node connections (see the Section called *Port Number for Node Connections*). By default, the port is not open.

Static nodes are started with the following command line:

```
enfnodeserver -b -d -n <root_host> <port>
```

This command line starts the node server as a background daemon in batch mode. The node server connects to the EnFuzion root on the **<root_host>** and **<port>**. If the connection with the root is terminated, the node server tries again.

The **-b** and **-n <root_host> <port>** options can be also specified in the **node.config** file on the node instead of on the command line:

```
connect on
roothost "<root_host>"
rootport "<port>"
batch on
```

Optionally, the **connectretry** and **connectdelay** options can be specified to change the EnFuzion provided default values.

EnFuzion provides a simple installation of the node server, so that it is started automatically at the computer boot time. For Windows, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 3. For Linux/Unix, check out the Section called *Starting EnFuzion Nodes at the Computer Boot Time* in Chapter 4.

Options in the **node.config** file are described in more detail in the Section called *Specifying Node Configuration Options* in Chapter 7.

Specifying Root Configuration Options

Root options are specified in an options file, called **root.options**. In addition to being specified in the options file, most root options can be accessed and modified through the EnFuzion API as part of a cluster object and through command line options. The name of the variable in the EnFuzion API is derived from its corresponding root configuration option by adding ENF prefix and using all capital letters. See the Section called *Application Programming Interface* in Chapter 10. See the Section called *The Dispatcher Options* in Chapter 9 for details on command line options. The rest of the section provides details about the **root.options** file.

The root.options File

EnFuzion checks for the **root.options** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

root.options contains lines with user defined option values. Lines that start with "#" are treated as comments.

The following sections describe configuration options in detail.

Specifying Available Third Party Software Licenses

If a commercial third party software is used to run programs on the cluster, often only a limited number of licenses might be available. This option specifies the number of available licenses in the EnFuzion cluster.

Available licenses are specified as:

```
licensepool   <name>=<value>   [, <name>=<value> ]
```

An example use is:

```
# specify available licenses for third party software
licensepool app1=5, app2=12
```

The example specifies 5 available licenses for the **app1** application and 12 available licenses for the **app2** application.

Each run can specify a set of license requirements for its jobs in the ENFLICENSES run options as described in the Section called *Run Options* in Chapter 8. Jobs are scheduled for execution by EnFuzion only if their required licenses are available.

Enforcing Privileges

This option specifies if the Dispatcher enforces user privileges or not. If the privilege enforcement in EnFuzion is turned on, regular EnFuzion users can only submit new runs and control their own runs. They are not allowed to control the cluster by performing actions, such as removing a run from a different user, adding and removing nodes, shutting down the cluster, and modifying cluster and node settings and properties. These restrictions do not apply to EnFuzion administrators, which are users specified in the **admins** file. By default, privileges are not enforced.

Privilege enforcement is specified as:

```
privileges on | off
```

Example:

```
# user privileges:
#     off - no restrictions, on - owner/admin capabilities only
privileges off
```

Rejecting Anonymous Run Submission

This option specifies if the Dispatcher allows users with anonymous ID to submit runs or not. **Anonymous** is a generic user ID, which is used by EnFuzion for users without an identification string. Most often, it is used for web based users. By default, anonymous users are allowed to submit runs.

Anonymous run submission is specified as:

```
noanonsubmit on | off
```

Example:

```
# disable submission of anonymous runs:
#     off - no restrictions, on - no anonymous submissions
noanonsubmit off
```

Prevent Execution of User Programs on the EnFuzion Root System

By default, user jobs can execute programs on the EnFuzion root host and access and modify files on the host. If required due to security reasons, this access can be limited. By setting the protect option to ’off’, user jobs will not be allowed to execute programs on the EnFuzion root host or access or modify files outside of their run directory.

Protection of the EnFuzion root host is specified as:

```
protect on | off
```

Example:

```
# prevent execution of user programs on the root system:
```

```
#     off - no restrictions, on - no execution on the root system
protect off
```

Port Number for the Eye

This option specifies the port number, which is used by users to connect to the Eye. The default port number is 10101.

Port number for the Eye is specified as:

```
eyeport <number>
```

Example:

```
# set the Eye port number, used for browser connections
eyeport 10101
```

Port Number for the HTTP Based Interface

This option specifies the port number of the HTTP based interface. This port is used by external applications to submit jobs to EnFuzion and retrieve results using the Internet HTTP protocol. By default, the port is not available. This option must be explicitly configured to enable the HTTP based API.

Port number for HTTP based API is specified as:

```
httpport <number>
```

Example:

```
# set the HTTP service port for submit clients
httpport 10108
```

The HTTP based interface is described in detail in the Section called *HTTP Based Application Programming Interface* in Chapter 10.

Port Number for Node Connections

This option specifies the port number that is used by nodes to connect to the root when they are started independently. By default, the port is not available unless it is explicitly configured.

Port number for node connections is specified as:

```
rootport <number>
```

Example:

```
# set the root port number, used for node connections
rootport 10103
```

If this option is enabled, then the Dispatcher broadcasts its host and port address on the local network once a minute. The broadcast can be disabled as described in the Section called *Port Number for Broadcasting the Address*.

Port Number for Broadcasting the Address

This option specifies the port number that is used by the root to broadcast its host and port address on the local network. This broadcast allows nodes to discover the root without being configured with any specific addresses.

The address broadcast is activated only when the port for node connections is enabled as described in the Section called *Port Number for Node Connections*. By default, the address is broadcast on port 10107 every minute. The broadcast is disabled, if the port number is -1.

Port number for broadcasting the address is specified as:

```
commport <number>
```

Example:

```
# set the broadcast port
commport 10107
```

The corresponding option on nodes is described in the Section called *Communication Port* in Chapter 7.

Port Number for Job Execution

This option specifies the port number that is used by user jobs on EnFuzion nodes to execute services, such as file copying or execution of commands, on the root. The default port value is dynamically assigned by the system.

Port number for job execution is specified as:

```
jobport <number>
```

Example:

```
# set the job port number, used for job connections from nodes
jobport 10104
```

Port Number for Node Starter Connections

This option specifies the port that the enfnodestarter program uses to accept node requests during the node start sequence. The default port value is dynamically assigned by the system.

Port number for node starter connections is specified as:

```
startport <number>
```

Example:

```
# set the start port number, used for starting nodes
startport 10105
```

Note: When this option is used, the concurrent node activations option **maxstart** must be set to 1. Otherwise, several nodes starters will attempt to use the same port, which will lead to significantly longer time to start nodes.

Queueing Policy

This option specifies the policy to execute runs that have the same priority level. By default, queueing is off and nodes are allocated to runs at the same priority level according to their priority weights. If queueing is turned on, then runs are placed in a queue and executed on a first come, first serve basis. Priority weights have no effect with queueing turned on, but priority levels are still enforced. Runs with a higher priority level get node allocations first. The default value for the queueing policy is off.

Queueing policy is specified as:

```
queue on | off
```

Example:

```
# set queueing policy: off - use priority weights, on - first come, first serve
queue off
```

Multiple Remote Nodes from One Host

This option determines whether multiple remote nodes are allowed to connect from a single host or not. By default, the option is off and only a single remote node is allowed to connect from a single host. When multiple nodes connect to the Root, the last connection will be kept active while all previous attempts will be closed. If this option is on, them all connections will be kept active, so that multiple nodes can execute on one host. The default value for the multiple remote nodes is off.

The multiple remote nodes option is specified as:

```
multinodes on | off
```

Example:

```
# allow multiple remote nodes from a single host:
#       off - one remote node only, on - multiple remote nodes from a host
multinodes off
```

Autonomous Node Operation

The **bind** option determines whether the node processes can continue to operate autonomously after the root connection is terminated. During the autonomous operation, the jobs on the node continue to execute and wait with results, the state on the hard disk is maintained and the node is trying to reconnect to the EnFuzion root. When the node successfully connects to the root, the results are transmitted and the node operation continues. By default, the connection with nodes is required all the time. In that case, if the connection with the node is lost, all its jobs are immediately rescheduled for execution.

The **bind** option is specified as:

```
bind on | off
```

Several options must be configured for the autonomous node operation to work. The requirements to configure the autonomous node operation are as follows. The **bind** option must be turned off on both the root and the node (see the Section called *Bind* in Chapter 7 for the node configuration). The **jobport** option must be defined on the root and must specify a fixed port (see the Section called *Port Number for Job Execution*). The connection between the root and the node must be initiated by the node, so the node must be either a dynamic or a static node (see the Section called *Nodes with No Root Control, Connection Initiated by the Node*).

Example:

```
# off - autonomous node operation, on - connected node operation, default on
bind on
```

Wait Limit

waitlimit specifies the time that the root waits for a node that operates in the autonomous mode. If the node does not connect to the root within this time, then the root reschedules its jobs for execution. If the **bind** option is turned on and autonomous operation is not permitted, this option has no effect.

The **waitlimit** option is specified as:

```
waitlimit <seconds>
```

Examples:

```
# waiting time for a disconnected node to connect, default infinite
waitlimit 86400
```

By default, there is no wait time limit and the root waits indefinitely for a node connection. The **waitlimit** option has a value of -1 in this case.

Deleting Obsolete User Directories

This option specifies the interval after which the run directories are considered obsolete and are deleted by EnFuzion. The default value is 7 days, which is 604800 seconds.

The interval to delete obsolete user directories is specified as:

```
cleanuplimit <seconds>
```

Example:

```
# delete obsolete run directories after run completion, in seconds, default 7 days
cleanuplimit 604800
```

Allowing Remote Access to the Dispatcher Interface

By default, it is possible to connect to the Dispatcher port and to use the **enfcmd** program from any computer. If required due to security reasons, this access can be limited. By setting the remoteaccess option to 'off', the Dispatcher port can be accessed only from the local computer.

Remote access is specified as:

```
remoteaccess on | off
```

Example:

```
# remote access to the Dispatcher API
#     off - local access only, on - no restrictions
#remoteaccess on
```

Restricting Access to the Dispatcher Interface

Allow and deny options control access to the Dispatcher interface from hosts on the network. The remoteaccess option must be turned on for these allow and deny options to have any effect.

Allow and deny options are specified as:

```
apiallow <address>
apideny <address>
```

The <address> parameter can be either a single IP address, like 192.168.11.100, or a network address, like 192.168.11.0/24, where 24 specifies valid bits in the address. This network address denotes all IP addresses in the form 192.168.11.<nnn>, where <nnn> can be any number between 0 and 255. Multiple allow and deny options may be included in the same **root.options** file.

If there are no allow and no deny options in the **root.options** file, all clients are allowed to connect. If there is at least one allow or deny option in the file, access is denied unless explicitly allowed by an option.

> **Note:** There are no special provisions for the local host address or for the 127.0.0.1 address. If access to the Dispatcher is restricted and access from the local host is required, then these addresses must be explicitly allowed. 127.0.0.1 must be available for the Eye to work. If the 127.0.0.1 address is disabled, the Eye will not work, since it uses this interface to communicate with the Dispatcher.

The list of allowed and denied entries can be obtained through the EnFuzion API. Variables ENFAPIALLOW and ENFAPIDENY contain allow or deny entries from **root.options**. Both allow and deny entries can be retrieved in the same order as entered in the **root.options** file through the API variable ENFAPIACCESS. These variables are read-only.

The authentication is done in the following manner. The IP address of the connecting client is matched against allow and deny options in the order in which they appear in the file. If the last option that matches the client IP address is allow, then the client is connected to the Dispatcher interface. Otherwise, the connection is denied.

Example:

```
# allow/deny Dispatcher API access from specific hosts/networks
apiallow 192.168.11.0/24
apideny  192.168.11.100
```

This example allows access to the Dispatcher API from any 192.168.11.<nnn> address, except 192.168.11.100.

Restricting Access to the HTTP based Interface

Allow and deny options control access to the HTTP based interface from hosts on the network. The HTTP interface must be enabled with the **httpport** option as described in the Section called *Port Number for the HTTP Based Interface*. If the **httpport** option is not enabled, allow and deny options have no effect.

Allow and deny options are specified as:

```
httpallow <address>
httpdeny <address>
```

The <address> parameter can be either a single IP address, like 192.168.11.100, or a network address, like 192.168.11.0/24, where 24 specifies valid bits in the address. This network address denotes all IP addresses in the form 192.168.11.<nnn>, where <nnn> can be any number between 0 and 255. Multiple allow and deny options may be included in the same **root.options** file.

If there are no allow and no deny options in the **root.options** file, all clients are allowed to use the HTTP interface. If there is at least one allow or deny option in the file, access is denied unless explicitly allowed by an option.

> **Note:** There are no special provisions for the local host address or for the 127.0.0.1 address. If access to the HTTP interface is restricted and access from the local host is required, then these addresses must be explicitly allowed.

The list of allowed and denied entries can be obtained through the EnFuzion API. Variables ENFHTTPALLOW and ENFHTTPDENY contain allow or deny entries from **root.options**. Both allow and deny entries can be retrieved in the same order as entered in the **root.options** file through the API variable ENFHTTPACCESS. These variables are read-only.

The authentication is done in the following manner. The IP address of the connecting client is matched against allow and deny options in the order in which they appear in the file. If the last option that matches the client IP address is allow, then the client is connected to the HTTP interface. Otherwise, the connection is denied.

Example:

```
# allow/deny access to the HTTP service from specific hosts/networks
httpallow 192.168.11.0/24
httpdeny  192.168.11.100
```

This example allows access to the HTTP interface from any 192.168.11.<nnn> address, except 192.168.11.100.

Restricting Node Access to the Dispatcher

Allow and deny options control access to the Dispatcher from nodes on the network. Only nodes that are connecting directly to the EnFuzion Root are effected. These options have no effect on nodes that are started by the Dispatcher.

Allow and deny options are specified as:

```
nodeallow <address>
nodedeny <address>
```

The <address> parameter can be either a single IP address, like 192.168.11.100, or a network address, like 192.168.11.0/24, where 24 specifies valid bits in the address. This network address denotes all IP addresses in the form 192.168.11.<nnn>, where <nnn> can be any number between 0 and 255. Multiple allow and deny directives may be included in the same **root.options** file.

If there are no allow and no deny options in the **root.options** file, all clients are allowed to connect. If there is at least one allow or deny option in the file, access is denied unless explicitly allowed by an option.

> **Note:** There are no special provisions for the local host address or for the 127.0.0.1 address. If node access is restricted and access from the local host is required, then these addresses must be explicitly allowed.

The list of allowed and denied entries can be obtained through the EnFuzion API. Variables ENFNODEALLOW and ENFNODEDENY contain allow or deny entries from **root.options**. Both allow and deny entries can be retrieved in the same order as entered in the **root.options** file through the API variable ENFNODEACCESS. These variables are read-only.

The authentication is done in the following manner. The IP address of the connecting node is matched against allow and deny options in the order in which they appear in the file. If the last option that matches the node IP address is allow, then the node is connected to the Dispatcher. Otherwise, the connection is denied.

Example:

```
# allow/deny nodes from specific hosts/networks
```

```
nodeallow 192.168.11.0/24
nodedeny  192.168.11.100
```

This example allows EnFuzion nodes from any 192.168.11.<nnn> address, except 192.168.11.100.

Restricting Access to the Eye

Allow and deny options control access to the Eye from hosts on the network.

Allow and deny options are specified as:

```
eyeallow <address>
eyedeny <address>
```

The <address> parameter can be either a single IP address, like 192.168.11.100, or a network address, like 192.168.11.0/24, where 24 specifies valid bits in the address. This network address denotes all IP addresses in the form 192.168.11.<nnn>, where <nnn> can be any number between 0 and 255. Multiple allow and deny directives may be included in the same **root.options** file.

If there are no allow and no deny options in the **root.options** file, all clients are allowed to connect. If there is at least one allow or deny option in the file, access is denied unless explicitly allowed by an option.

> **Note:** There are no special provisions for the local host address or for the 127.0.0.1 address. If access to the Dispatcher is restricted and access from the local host is required, then these addresses must be explicitly allowed.

The authentication is done in the following manner. The IP address of the connecting client is matched against allow and deny options in the order in which they appear in the file. If the last option that matches the client IP address is allow, then the client is connected to the Eye. Otherwise, the connection is denied.

Example:

```
# allow/deny Eye access from specific hosts/networks
eyeallow 192.168.11.0/24
eyedeny  192.168.11.100
```

This example allows EnFuzion nodes from any 192.168.11.<nnn> address, except 192.168.11.100.

Starting the Eye

This options specifies if the Dispatcher starts the Eye which provides web based user interface, at its startup time. By default, the Eye is started by the Dispatcher.

Starting the Eye is specified as:

```
eyestart on | off
```

Example:

```
# start of the Eye by the Dispatcher
eyestart on
```

Terminating the Eye

This options specifies if the Dispatcher terminates the Eye which provides a web based user interface, at its termination time. By default, the Eye is terminated by the Dispatcher, if the Dispatcher is executed in the single run mode and not terminated if the Dispatcher is executed in the multiple run mode.

Terminating the Eye is specified as:

```
eyeterminate on | off
```

Example:

```
# termination of the Eye by the Dispatcher
eyeterminate off
```

Off Periods

Off periods prevent the execution of EnFuzion jobs during the time specified. By default, EnFuzion jobs can run at any time. During off periods, all EnFuzion node processes under the Dispatcher control are terminated. The processes are started again by the root after the off period expires.

Off periods are specified as:

```
off [ day <day>[-<day>] ] [ time <time>-<time> ]
off date yyyy/mm/dd [ time <time>-<time> ]
```

For details on the time and date format see the Section called *Specifying Time Interval* in Chapter 7 and the Section called *Specifying Days, Months, Date* in Chapter 7.

Examples:

```
# interval without job execution
off day Mon-Fri time 7:30-17:30
```

The example above prevents EnFuzion job execution from Monday to Friday between 7:30 and 17:30.

Specifying Mail Server System

This option specifies the mail server system that is used by EnFuzion to send electronic messages. These can be requested by users to receive a notification of certain events, such as a run start, end or abort.

On Linux/Unix systems, the default value is to use the local mail program. On Windows system, this option must be always specified.

An mail server is specified as:

```
mailserver "<host_name>"
```

<host_name> must be an active SMTP server, configured to forward messages.

An example use is:

```
# mail server host name
mailserver "mail.company.com"
```

Specifying Mail Service Port

This option specifies the service port provided by the SMTP server. The default value is the standard SMTP service port 25.

The SMTP service port is specified as:

```
mailport <number>
```

An example use is:

```
# mail server port number
mailport 25
```

Specifying Mail Sender

This option specifies the sender of electronic messages sent by the EnFuzion root. The default value is the user that is executing EnFuzion programs on the root host and the root host name.

An mail sender is specified as:

```
mailuser "<user_name>@<host_name>"
```

An example use is:

```
# mail user From: identity
mailuser "enfuzion@company.com"
```

Concurrent Node Activations

This option limits the number of concurrent node activations. If an EnFuzion configuration consists of a large number of nodes and all the nodes were to be activated at the same time, it could overload the root computer. Therefore, EnFuzion limits concurrent node activations. If no value is specified, the limit is set to 32.

Concurrent node activations are specified as:

```
maxstart <number>
```

Example:

```
# number of concurrent node activations, default 32
maxstart 32
```

Node Restart Period

If a node or a network connection to the node fails and the node is declared as 'down', EnFuzion tries to restart the node after a certain period of time. This options specifies the wait period before restarting a down node. If no value is specified, the default restart period is 15 minutes.

The node restart period is specified as:

```
restart <time>
```

Example:

```
# delay time to restart down nodes, default 15 minutes
restart 00:15:00
```

Heartbeat Period

This option specifies the interval for heartbeats between the root and the node machines. The default value is 300 seconds.

Heartbeat period is specified as:

```
heartbeat <seconds>
```

Example:

```
# node heartbeat interval, in seconds, default 300s
heartbeat 300
```

Disconnect Period

This option specifies the period that either a root or a node machine waits for a heartbeat signal. If no heartbeat signal is detected within this period, the connection is assumed dead and the node is terminated. The default value is 480 seconds.

Disconnect period is specified as:

```
disconnect <seconds>
```

Example:

```
# interval without heartbeat for declaring a node down, default 480s
disconnect 480
```

Minimum Time to Obtain Resource Information

Nodes are repeatedly sending reports on resource consumption of executing jobs to the root. This option specifies the minimum time between two messages from one node. The actual time might be larger, but it will not be smaller than the value of this option. The default value is 15 seconds. The default value can be increased, if the network load is too high. A higher value reduces the network load at the expense of resource consumption being sampled less often. A lower value than 15 seconds has no effect, since nodes are collecting resource consumption every 15 seconds. This option is relevant only to Windows nodes, since information on resource consumption is collected only on Windows.

Resource time is specified as:

```
resources <seconds>
```

Example:

```
# time interval to obtain resource reports from nodes
resources 15
```

Complete Logs

All EnFuzion events are recorded in the **enfuzion.log** file. To enhance performance, run specific events, which can generate a large number of log entires, can be disabled. In that case, only cluster and node events are recorded in the log file. Run related events are recorded only in run specific logs. The default value for the completelogs root option is off.

Complete logs are specified as:

```
completelogs on | off
```

Example:

```
# exclude run, job and datajob events from the enfuzion.log file
completelogs off
```

Maximum Dispatcher Log Size

The maximum size of Dispatcher log file is limited by logsizelimit root option, with units in Mb. Whenever a log grows over its limit, it is renamed to "enfuzion-%d.log", where %d is the smallest integer with a nonexistent file. A new log file is started in **enfuzion.log**. The default value for logsizelimit is 10Mb.

Maximum Dispatcher log size is specified as:

```
logsizelimit <number>
```

Example:

```
# enfuzion.log size for file rotation, in Mb, default 10Mb
logsizelimit 10
```

Maximum Datastream Job Size

The maximum datajob size is limited to enhance security. The default size can be changed through the maxdatastream option, with units in Kb. The default value for maxdatastream is 20Kb.

Maximum datastream job size is specified as:

```
maxdatastream <number>
```

Example:

```
# maximum datajob size, in Kb, default 20Kb
maxdatastream 20
```

Sample root.options File

The following is a sample **root.options** file. All options in the file are disabled with comments.

To change default root option values, store the text below to file **root.options** in the EnFuzion **config** subdirectory and modify option values for your environment.

```
#--- store the text below to root.options ---
#
# EnFuzion Root Configuration File
#       this is only a sample
#       uncomment and modify lines for your configuration

# specify available licenses for third party software
#licensepool app1=5, app2=12

# user privileges:
#     off - no restrictions, on - owner/admin capabilities only
#privileges off
# disable submission of anonymous runs:
#     off - no restrictions, on - no anonymous submissions
#noanonsubmit off
# prevent execution of user programs on the root system:
#     off - no restrictions, on - no execution on the root system
#protect off

# set the Eye port number, used for browser connections
#eyeport 10101
# set the root port number, used for node connections
#rootport 10103
# set the job port number, used for job connections from nodes
```

```
#jobport 10104
# set the start port number, used for starting nodes
#startport 10105
# set the broadcast port
#commport 10107
# HTTP service port for submit clients
#httpport 10108

# set queueing policy: off - use priority weights, on - first come, first serve
#queue off

# allow multiple remote nodes from a single host:
#       off - one remote node only, on - multiple remote nodes from a host
#multinodes off

# off - autonomous node operation, on - connected node operation, default on
#bind on
# waiting time for a disconnected node to connect, in seconds, default infinite
#waitlimit 86400

# delete obsolete run directories after run completion, in seconds, default 7 days
#cleanuplimit 604800

# remote access to the Dispatcher API
#remoteaccess on
# allow/deny Dispatcher API access from specific hosts/networks
#apiallow 192.168.11.0/24
#apideny  192.168.11.100
# if any apiallow, apideny is present, enable the line below for the Eye to work
#apiallow 127.0.0.1

# allow/deny nodes from specific hosts/networks
#nodeallow 192.168.11.0/24
#nodedeny  192.168.11.100

# allow/deny access to the HTTP service from specific hosts/networks
#httpallow 192.168.11.0/24
#httpdeny  192.168.11.100

# allow/deny Eye access from specific hosts/networks
#eyeallow 192.168.11.0/24
#eyedeny  192.168.11.100
# if any eyeallow, eyedeny is present, enable the line below for local access
#eyeallow 127.0.0.1

# start of the Eye by the Dispatcher
#eyestart on
# termination of the Eye by the Dispatcher
#eyeterminate off

# interval without job execution
#off day Mon-Fri time 7:30-17:30
```

```
# mail server host name
#mailserver "mail.company.com"
# mail server port number
#mailport 25
# mail user From: identity for outgoing notices
#mailuser "enfuzion@company.com"

# number of concurrent node activations, default 32
#maxstart 32
# delay time to restart down nodes, default 15 minutes
#restart 00:15:00

# node heartbeat interval, in seconds, default 300s
#heartbeat 300
# interval without heartbeat for declaring a node down, default 480s
#disconnect 480

# minimum interval to obtain resources from a node, default 15s
#resources 15

# exclude run, job and datajob events from the enfuzion.log file
#completelogs off
# enfuzion.log size for file rotation, in Mb, default 10Mb
#logsizelimit 10

# maximum datajob size, in Kb, default 20Kb
#maxdatastream 20

#-------------------------------------------------------------
```

Specifying User Identities

EnFuzion supports the concept of a user. All interactions with EnFuzion at run time are assigned an owner user ID. This owner assignment is used in accounting reports to identify the work done by a single user or to restrict permitted user actions.

A user ID is a string in the form **<user>@<domain>**. By default, **<user>** is the account name of the user that is submitting the run and **<domain>** is the host name of the computer where the submission is performed. If EnFuzion is unable to determine the default user string, a generic **anonymous** user ID is assigned as the run owner.

This default user assignment can be changed through the **users** configuration file. This capability of changing the default user assignment is useful when a single EnFuzion user uses several systems or even several user accounts to interact with EnFuzion.

As an example, suppose that EnFuzion is used by Bob from the QA department, who regularly uses multiple computers: his desktop, his laptop and Jane's desktop computer. By default, EnFuzion will assign a different user to Bob on each machine, resulting in three users: **bob@bobsdesktop.qa.company.com**, **bob@bobslaptop.qa.company.com**, and

bob@janesdesktop.qa.company.com. Instead of three users, the administrator wants to identify Bob as one EnFuzion user **bob@qa.company.com**.

The following section specifies how the default user assignment can be changed through the **users** configuration file.

The users File

EnFuzion checks for the **users** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

The **users** file consists of mapping rules, one mapping rule per line. Lines that start with "#" are treated as comments and ignored.

A mapping rule is described as:

```
<template> [ , <template> ]    user <result>
```

The default user assignment is taken as a starting string. If any of the templates in a line match the string, then the string is replaced with the <result> in that line. The process is repeated until there are no matches or changes in the string. The final string is the assigned user.

A <template> can be one of three forms: **<account>@<host>**, which matches the user and the host; **<account>** with no '@' character, which matches only the user; **@<host>** which starts with '@' character and matches only the host.

A <template> can contain wild card constructs: '*', '?' and '[...]'. '*' matches any number of characters, '?' matches one character, and '[...]' matches any of the characters inside the square brackets. '[...]' can contain a range, specified with '-'. An example is '[a-c]'.

A <result> syntax is similar to a <template>: **<account>@<host>** rewrites the user and the host; **<account>** with no '@' character rewrites only the user; **@<host>** which starts with '@' character rewrites only the host. Wild card constructs are not allowed in the <result>.

Here are some examples of how mapping rules are applied.

The problem with Bob, which requires access from three different systems, is solved with the following mapping rule:

```
bob@bobsdesktop.qa.company.com, \
bob@janesdesktop.qa.company.com, \
bob@bobslaptop.qa.company.com    user  bob@qa.company.com
```

This rule maps the user Bob from all three systems to one EnFuzion user. Note that '\' may be used to split a single logical line over several lines in the file in order to improve readability.

The following rule allows Bob to use EnFuzion from any computer in the QA department:

```
bob@*.qa.company.com    user  @qa.company.com
```

The following rule allows all users from all systems in the QA department to use EnFuzion:

```
@*.qa.company.com    user  @qa.company.com
```

The following rule maps all users from the QA department to a single EnFuzion user:

```
@*.qa.company.com    user  qa@company.com
```

The following rule maps all root and Administrator accounts to the enfuzion-admin user, which can be then given EnFuzion administrative rights:

```
root, Administrator    user  enfuzion-admin@company.com
```

Specifying Groups

EnFuzion users can be grouped by the administrator in order to report combined activities of related users. Users can be members of one or more user groups.

Groups are useful to generate combined activity reports for different departments or group projects.

Groups are specified in the **groups** file, which is described in the next section.

The groups File

EnFuzion checks for the **groups** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

The **groups** file consists of group lists, one group list per line. Lines that start with "#" are treated as comments and ignored.

A group list is described as:

```
<group_name>    <user_name>  [ , <user_name> ]
```

The following example describes a group QA with members from the QA department:

```
QA    bob@qa.company.com, jane@qa.company.com, \
      john@qa.company.com, june@qa.company.com
```

Specifying Administrators

EnFuzion administrator are users that can perform any action without restrictions.

If the privilege enforcement in EnFuzion is turned on, regular EnFuzion users are not allowed to control the cluster by performing actions, such as removing a run owned by a different user, adding and removing nodes, shutting down the cluster, and modifying cluster and node settings and properties. These restrictions do not apply to EnFuzion administrators, which are users specified in the **admins** file.

The admins File

EnFuzion checks for the **admins** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

The **admins** file consists of a list of EnFuzion users, one user per line. Lines that start with "#" are treated as comments and ignored.

The following example gives EnFuzion administrative rights to the **enfuzion-admin@company.com** user:

```
enfuzion-admin@company.com
```

Specifying User Accounts for Job Execution on Nodes

By default, all EnFuzion programs on the node are executed under a single user account. For Linux/Unix nodes, EnFuzion can be configured to allow EnFuzion users to select a different account to execute their programs.

The node account to execute user jobs is determined individually for each run and node pair. The EnFuzion root must explicitly allow user accounts on nodes with user rules in the **user.accounts** file. If no user rules are configured on the EnFuzion root, which means that the **user.accounts** file is empty or does not exist, then the default EnFuzion account is being used. If there are user rules in the **user.accounts** file, then these rules are evaluated and the resulting user account is requested on the node.

The EnFuzion node also needs to be configured for user accounts as described in detail in the Section called *Specifying Node User Accounts* in Chapter 7. Since EnFuzion node programs execute under a

regular user account by default, they must be configured to be able to execute programs under a different user. If the node is not configured for user accounts, then the request is rejected.

The following section provides details on user rules in the **user.accounts** file.

The user.accounts File

EnFuzion checks for the **user.accounts** file in the following locations: the local working directory, the **$ENFUZION_PATH/config** directory, and the **config** subdirectory of the EnFuzion installation directory.

On Linux/Unix, the default installation directory is **$HOME/enfuzion** for regular users and **/usr/local/enfuzion** for the root user. Both locations are checked.

On Windows NT/2000/XP, the default installation directory is **C:\enfuzion**.

The **user.accounts** file contains user rules for user accounts on EnFuzion nodes, one rule per line. Lines that start with "#" are treated as comments and ignored.

A user rule is described as:

```
[user <template> [,<template>]]   [host <host> [,<host>]]  \
                      account <account>[,<account>]
```

A <template> can be one of three forms: **<account>@<host>**, which matches the user and the host; **<account>** with no ’@’ character, which matches only the user; **@<host>** which starts with ’@’ character and matches only the host.

<template> and <host> can contain wild card constructs: ’*’, ’?’ and ’[...]’. ’*’ matches any number of characters, ’?’ matches one character, and ’[...]’ matches any of the characters inside the square brackets. ’[...]’ can contain a range, specified with ’-’. An example is ’[a-c]’.

User rules are evaluated a follows:

- lines in "user.accounts" are processed one by one;
- if a line matches the run owner user ID and the node, then the rule is applied to produce a node user account
- if none of the lines matches the run owner and the node, the default EnFuzion account is being used.

The line matching with the run owner and the node is performed as follows:

- if the run owner user ID matches one of the users under the **user** component, then the user component matches
- if the **user** component is not present, then it matches by default
- if the node host matches one of the hosts under the **host** component, then the host component matches
- if the **host** component is not present, then it matches by default
- if the **user** component and the host component match, then the line matches

When the run owner user ID and the node match a rule, then the rule is applied to determine the node user account. The node user account is determined from an optional user account, requested by the run user and from accounts specified in the rule. The node user account is determined as follows:

- if the run user requests a node user name and that user name matches one of the accounts under the **account** component, then that node user name is selected;
- if the run user does not request a node user name or if the user name requested does not match any of the accounts under the **account** component, then the first account in the **account** component is selected;

EnFuzion predefines some system node user names. These are: **__deny__**, which denies node access to the user; **__default__**, which specifies the default EnFuzion account on the node; and **__user__**, which equals the node user name to the user account name of the run owner user ID.

Here are some examples of rules for assigning node user accounts.

To use the default EnFuzion account on all nodes, **user.accounts** must be empty or non-existent.

To use the user name of the run owner for job execution, **user.accounts** contains the following line:

```
account __user__
```

EnFuzion users can select to use their own account or a generic **enfuzion** account:

```
account __user__,enfuzion
```

Root Based Security Features

This section provides details how to avoid clear text passwords in the **enfuzion.nodes** file.

Encrypted Passwords in enfuzion.nodes

EnFuzion nodes on remote hosts are specified in the file **enfuzion.nodes**. Normally, each node is described with a line, containing a user account and a password in clear text. If clear text passwords in the **enfuzion.nodes** file are not acceptable for security reasons, they can be encrypted with the **Enfprotectpass** utility. The utility is part of the EnFuzion package.

The Enfprotectpass utility takes the file **enfuzion.nodes** in the current directory and produces a file with encrypted user accounts and passwords. The output file is named **enfuzion.nodes.e**. User accounts are replaced with "*" and passwords are replaced with a field, containing encrypted user accounts and passwords. The field starts with "***". Clear text passwords in the original configuration file can be changed to encrypted fields either by renaming the entire **enfuzion.nodes.e** file to **enfuzion.nodes** or by

manually replacing clear text passwords with the corresponding encrypted fields. The default input and output file names can be changed through command line arguments.

The Enfprotectpass has the following command line arguments:

```
enfprotectpass \
    [ -v ] \
    [ -d ] \
    [ -i <file_name> ] \
    [ -o <file_name> ] \
    [ -s ]
```

- **-v**

  Print out the program version and argument descriptions.

- **-d**

  Read input from the standard input instead of the **enfuzion.nodes** file.

- **-i <file_name>**

  Read input from the file <file_name> instead of the **enfuzion.nodes** file.

- **-o <file_name>**

  Write output to the file <file_name> instead of to the **enfuzion.nodes.e** file.

- **-s**

  Write output to the standard output instead of to the **enfuzion.nodes.e** file.

Example:

```
enfprotectpass -o enfuzion.out
```

This option creates file **enfuzion.out**, which contains encrypted user information from the **enfuzion.nodes** file. The output file can be renamed **enfuzion.nodes** and used instead of the original **enfuzion.nodes** file.

> **Note:** Encrypted fields contain the user account and password information. Whenever a user name or a password on a node changes, the encrypted field for that node must be generated again.

EnFuzion installations that require a different method of password encryption than the one provided by EnFuzion can use user defined methods as described in the Section called *User Defined Decryption Primitives* in Chapter 7.

Chapter 7. Node Configuration

This chapter provides details about EnFuzion node configuration.

In most environments, no changes to the default configuration of EnFuzion nodes are required, once the EnFuzion node software is installed. However, the default EnFuzion behavior can be changed through node configuration options.

An EnFuzion node possesses several configuration options. These options can be tuned for specific user environments, in order to improve performance or to manage security aspects of EnFuzion operation. These options are provided in the **node.config** file. This file is optional for running EnFuzion and can contain only options that are relevant for a particular EnFuzion installation.

Another group of options consists of load monitoring settings. These options are separate from the node configuration options and can be activated, so that EnFuzion jobs execute only when a node is idle and not used for any other task. They determine when a node is available for EnFuzion jobs, so that they do not interfere with normal use of the node. The load monitoring options are specified in the **enfuzion.options** file.

The EnFuzion node software provides additional security features that enhance system provided security. These features include trusted hosts and executables, user defined decryption, and root authentication.

The rest of this chapter describes details about configuring node options, load monitoring options, and node based security features.

Specifying Node User Accounts

EnFuzion users can require that their programs are executed under a different user account than the default EnFuzion account on the node. This capability is supported only for Linux/Unix nodes.

Since EnFuzion node programs by default execute only under the EnFuzion account, they must be enabled to execute programs under a different user. If a node is not enabled for user accounts, then user requests to change the account are rejected.

To enable the execution of programs under user accounts, the EnFuzion **enfjobserver** program must be given additional permissions not granted during the standard EnFuzion node installation process. These permissions include the root account ownership and the permission to change the program owner. The permissions are granted with the following commands, which must be executed under the root user:

```
chown root enfjobserver
chgrp root enfjobserver
chmod +s enfjobserver
```

The commands above enable the node to allow execution under the user specified accounts. An additional configuration is required on the EnFuzion root as described in the Section called *Specifying User Accounts for Job Execution on Nodes* in Chapter 6.

Specifying Node Configuration Options

Node configuration options are specified in an options file, called **node.config**.

The node configuration file is read when the node server is started. If any of the node configuration options are changed, then the node server must be terminated and restarted for any changes to take the effect.

The rest of this section provides details about the **node.config** file.

The node.config File

EnFuzion checks for the **node.config** file in the following locations: the local working directory, the directory specified in the ENFNODE_PATH environment variable, and the main EnFuzion installation directory on the node. By default, the **node.config** file is located in the main EnFuzion installation directory on the node.

On Linux/Unix, the default location of the main EnFuzion installation directory is **$HOME/enfuzion**.

On Windows, the default location of the main EnFuzion installation directory is **C:\enfuzion**.

The **node.config** file contains lines with user defined option values. Lines that start with "#" are treated as comments.

The following sections describe configuration options in detail.

Requested Concurrent Jobs

The **joblimit** option specifies the maximum number of concurrent executing jobs on this node. Usually, its value is equal to the number of processors on the host.

The option is specified as:

```
joblimit <integer>
```

Examples:

```
# the number of concurrent jobs
joblimit 1
```

The default value is set to the number of CPUs on Windows, Linux, Mac OS X and Solaris operating systems and to 1 on all other systems.

There is an equivalent option in **enfuzion.options**. If both are specified, the value in **enfuzion.options** takes precedence. The **joblimit** option in the **node.config** file is the recommended use of the option. More information about **enfuzion.options** can be found in the Section called *Specifying Load Monitoring Options*.

Node Port

By default, when the node starts, it opens a port for the root to connect to. This port is assigned dynamically. If the network traffic between the root and the node is controlled by a firewall, then dynamic port assignment might not be compatible with the firewall. In such cases, the node can be configured with a static port.

The **nodeport** option is specified as:

```
nodeport <port_number>
```

Examples:

```
# set node port number, used for root connections
nodeport 10106
```

There is no default value for the **nodeport** option. If the option is not specified, the node port is assigned dynamically.

Connect

The **connect** option determines whether the node initiates the connection to the root. By default, the root connects to the node as specified in the **enfuzion.nodes** file. This default behavior can be changed using the **connect** option. If the option value is on, the node will connect to the root, and there is no need to specify the node in the **enfuzion.nodes** file on the root. However, the port on the root must be enabled in **root.options** as described in the Section called *Port Number for Node Connections* in Chapter 6.

The **connect** option is specified as:

```
connect on | off
```

Examples:

```
# set connect from the node: off - root connect, on - node connect
connect on
```

The default value for the **connect** option is off, meaning the root makes the connection to a node.

Communication Port

The node expects an announcement of the root host address and its port number on this port. When the **connect** option is on, the node requires the root host and its port number to connect to. If these values are not specified or if their value is 0, then the node waits on this port number to receive the root host address and the port to connect to. If the value is not specified, it waits on port 10107 by default.

The **commport** option is specified as:

```
commport <port_number>
```

Examples:

```
# the communication port number
commport 10107
```

The corresponding option on the root is described in the Section called *Port Number for Broadcasting the Address* in Chapter 6.

Connect Host

If the **connect** option value is on, meaning the node connects to the root, then this option can provide the root host name.

The **roothost** option is specified as:

```
roothost "<host_name>"
```

Examples:

```
# the root host to connect to
roothost "enfuzion.domain.com"
```

Note: <host_name> must be included in double quotes.

There is no default value for the **roothost** option. If the **connect** option is on and the **roothost** value is not specified, the node waits for a broadcast of the root address.

Connect Port

If the **connect** option is on, and the node connects to the root, then this option provides the port number for connection to the root host.

The **rootport** option is specified as:

```
rootport <port_number>
```

Examples:

```
# the root port number
rootport 10103
```

There is no default value for the **rootport** option.

Connect Backup Host

This option provides a backup value for the **roothost** option. The backup host is tried if the connection to the primary host fails.

The **backuphost** option is specified as:

```
backuphost "<host_name>"
```

Examples:

```
# the backup root host to connect to
backuphost "enfuzion1.domain.com"
```

Note: <host_name> must be included in double quotes.

There is no default value for the **backuphost** option.

Connect Backup Port

This option provides a backup value for the **rootport** option. The backup host and port are tried if the connection to the primary host and port fails.

The **backupport** option is specified as:

```
backupport <port_number>
```

Examples:

```
# the backup root port number
backupport 10103
```

There is no default value for the **backupport** option.

Connect Retry

If the **connect** option is on, and the node connects to the root, then this option specifies how many times the node tries to connect to the root. This option is useful when the root is not executing at all times, and nodes must wait for the root to become operational.

The **connectretry** option is specified as:

```
connectretry <number_of_tries>
```

If <number_of_tries> is 0, then the node tries to connect infinitely.

Examples:

```
# the number of tries to connect to the root, default 0, meaning infinite
connectretry 0
```

The default value for the **connectretry** option is 0, which means try infinitely.

Connect Delay

If the **connect** option is on, and the node connects to the root, then this option specifies the delay between attempts to connect to the root. This option is useful when the root is not executing at all times, and nodes must wait for the root to become operational.

The **connectdelay** option is specified as:

```
connectdelay <seconds>
```

Examples:

```
# the delay between different tries, in seconds, default 60s
connectdelay 60
```

The default value for the **connectdelay** option is 60s, which means the delay between connections is 1 minute.

Execution Time Limit

timelimit limits the node server execution time. After the node server exceeds the execution time limit, no new jobs are requested and the server terminates after all the jobs on the node complete. This option is useful when the node server execution is controlled by other job schedulers.

The **timelimit** option is specified as:

```
timelimit <seconds>
```

Examples:

```
# set the execution time limit to 100s
timelimit 100
```

By default, there is no execution time limit. The **timelimit** option has a value of -1 in this case.

Batch

The **batch** option determines whether the node process terminates after the root connection is terminated or not. By default, the node processes perform a cleanup on the node and terminate after the connection with the root is terminated. This default behavior can be changed with the **batch** option. If the option is on, the node does not terminate after the root termination, instead it continues to wait for another root connection.

The **batch** option is specified as:

```
batch on | off
```

Examples:

```
# batch mode: off - exit after the first connection, on - keep executing
batch off
```

The default value for the **batch** option is off. The node processes perform a cleanup and terminate after the root connection is terminated.

Bind

The **bind** option determines whether the node process can continue to operate autonomously after the root connection is terminated. During the autonomous operation, the jobs on the node continue to execute and wait with results, the state on the hard disk is maintained and the node is trying to reconnect to the EnFuzion root. When the node successfully connects to the root, the results are transmitted and the node operation continues. By default, the connection with the root is required all the time. In that case, if the connection with the root is lost, the node processes immediately perform a cleanup and terminate all jobs.

The **bind** option is specified as:

```
bind on | off
```

Several options must be configured for the autonomous node operation to work. The requirements to configure the autonomous node operation are as follows. The **bind** option must be turned off on both the root (see the Section called *Autonomous Node Operation* in Chapter 6 for the root configuration) and the node. The **jobport** option must be defined on the root and must specify a fixed port (see the Section called *Port Number for Job Execution* in Chapter 6). The connection between the root and the node must be initiated by the node, so the node must be either a dynamic or a static node (see the Section called *Nodes with No Root Control, Connection Initiated by the Node* in Chapter 6).

Examples:

```
# off - autonomous operation, on - connected operation, default on
bind on
```

The default value for the **bind** option is off. The node processes perform a cleanup and terminate after the root connection is terminated.

Wait Limit

waitlimit limits the time that the node continues to operate in the autonomous mode. If the node is unable to connect to the root within this time, then the node performs a cleanup and terminates all the jobs. If the **bind** option is turned on and autonomous operation is not permitted, this option has no effect.

The **waitlimit** option is specified as:

```
waitlimit <seconds>
```

Examples:

```
# waiting time for a disconnected node to connect, default infinite
waitlimit 86400
```

By default, there is no wait time limit and the node tries indefinitely to connect to the root. The **waitlimit** option has a value of -1 in this case.

Node Port Message

After the node is started, and if the **connect** option is off, a port is opened for the root to connect to. This option specifies whether the node port is reported to the program that started the node or not. This option is used primarily for internal EnFuzion purposes.

The node port message option is specified as:

```
report on | off
```

Examples:

```
# report the port number (internal use)
report on
```

The default value for the **report** option depends on the value of the connect option. If **connect** is on, then the node port message option is off. Otherwise, the default is on.

Hello Message

After the node is started, the node can exchange an initial sequence with the program that started it. This option specifies whether the initial sequence with the start program is exchanged or not. This option is used primarily for internal EnFuzion purposes.

The **hello** option is specified as:

```
hello on | off
```

Examples:

```
# exchange an initial sequence (internal use)
hello on
```

The default value for the **hello** option depends on the value of the **connect** option. If **connect** is on, then the **hello** option is off. Otherwise, the default is on.

Sample node.config File

The following is a sample **node.config** file. All options are disabled with comments.

To change default configuration values, store the text below to file **node.config** in the EnFuzion directory on the node and modify option values for your environment.

```
#------------ store the text below to node.config -------------
#
# EnFuzion Node Configuration File
#       this is only a sample
#       uncomment and modify lines for your configuration

# the number of concurrent jobs
#joblimit 1
```

```
# set node port number, used for root connections
#nodeport 10106

# set connect from the node: off - root connect, on - node connect
#connect on
# communications port to obtain the root service address
#commport 10107
# the root host to connect to
#roothost "enfuzion.domain.com"
# the root port number
#rootport 10103

# the backup root host to connect to
#backuphost "enfuzion1.domain.com"
# the backup root port number
#backupport 10103

# the number of tries to connect to the root, default 0, meaning infinite
#connectretry 0
# the delay between different tries, in seconds, default 60s
#connectdelay 60

# batch mode: off - exit after the first connection, on - keep executing
#batch off

# off - autonomous operation, on - connected operation, default on
#bind on
# waiting time for a disconnected node to connect, default infinite
#waitlimit 86400

# timelimit: execution time limit in seconds, default -1, meaning infinite
#timelimit 300

# report the port number (internal use)
#report on
# exchange an initial sequence (internal use)
#hello on
#-------------------------------------------------------------
```

Specifying Load Monitoring Options

Load monitoring options are specified in the **enfuzion.options** file. They provide load monitoring and resource sharing capabilities. The primary purpose of these options is to specify, whether the node is idle and available to execute EnFuzion jobs or busy with other tasks, which are not EnFuzion related. Load monitoring options can also be used to ensure that required resources are available on the node, such as memory and disk space.

Load monitoring options allow EnFuzion to utilize idle compute cycles, while systems are still fully available for their regular use. EnFuzion users benefit from maximum performance, while overloading of

EnFuzion nodes with jobs is prevented. EnFuzion respects the ownership of a computer and gives priority to interactive console users.

All options can be changed dynamically at run time. They are updated automatically at regular intervals every two minutes.

The rest of the section provides details about the **enfuzion.options** file.

The enfuzion.options File

Node options are specified in the option file called **enfuzion.options**. Several **enfuzion.options** files can be provided on a node. EnFuzion provides a system file, a local user file and run specific files.

The rest of this section explains file locations and syntax, and describes options in detail.

System and Local User Options

Each EnFuzion node can have its own system option file. In addition, each EnFuzion account on the node can have a local option file.

The system option file provides default values for the system. On Linux/Unix nodes, the system option file is located in **/var/opt/enfuzion/enfuzion.options**. On Windows NT/2000/XP nodes, the option file is stored in the main EnFuzion installation directory on the node. The default location is **C:\enfuzion\enfuzion.options**.

Users can change system values with a local options file within the limits of their user status. Users can be console users or remote users. The user at the console has complete control over the system and can change any of the default system values.

Remote users can change only those values that are not specified by the system. Required disk space, required main memory, requested maximum number of jobs, node directory and the termination signal can be changed by remote users at any time, because these options do not affect other users.

EnFuzion checks for the local user specific **enfuzion.options** file in the following locations: the local working directory, the directory specified in the ENFNODE_PATH environment variable, and in the main EnFuzion installation directory on the node.

The main EnFuzion installation directory is checked only on Linux/Unix, where the default location is **$HOME/enfuzion**.

The next section describes how to dynamically provide a local **enfuzion.options** file in the working directory.

Run Specific Options

The **enfuzion.options** file can be copied from the root machine to the node machine within a run file. This is useful for an option file that is run and application specific.

The copied file overrides local options and is valid only for the current EnFuzion run. It does not affect options for other runs.

The following command in the nodestart task copies the option file named **options** from the current working directory on the EnFuzion root to an EnFuzion node:

```
copy   ../options   node:../enfuzion.options
```

The following command in the main task copies the option file named options from the current working directory on the EnFuzion root to an EnFuzion node:

```
copy   ../options   node:../../enfuzion.options
```

The commands for the nodestart task and the main task are not the same, because these tasks execute in different directories.

In both cases, the run specific option file overrides a default local option file. By default, it takes about two minutes after **enfuzion.options** is copied for the new options to take effect. If it is necessary that option values become valid immediately, then the copy commands above must be followed by an options command. See the Section called *Command **copy*** in Chapter 8 and the Section called *Command **options*** in Chapter 8 for details.

File Syntax

The **enfuzion.options** file contains lines with user defined option values. Lines that start with "#" are treated as comments.

The following sections describe syntax for common elements.

Specifying Time Interval

Time is specified in one of the following three forms:

```
hh          hours
hh:mm       hours, minutes
hh:mm:ss    hours, minutes, seconds
```

Specifying Days, Months, Date

Days are specified as:

```
Sun, Mon, Tue, Wed, Thu, Fri, Sat
```

Months are specified as numbers or names:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
```

A date is specified as year, month, day:

```
yyyy/mm/dd      year, month, day
```

Conditional Options

Each option can be preceded by a condition. If the condition is true, then the option is used. If the condition is false, then the option is ignored. This functionality is useful, when the same **enfuzion.options** file is shared among multiple machines, but different option values are required. EnFuzion implements conditions based on the host name and on the existence of a file path.

The **host** condition lists valid hosts for a particular option. If one of the hosts in the list matches the local host, then the option is enforced. The syntax of the **host** condition is:

```
host "<host>"   <option>
```

More than one host name can be also specified, provided that host names are separated by commas:

```
host "<host_1>", ..., "<host_n>"   <option>
```

The <option> is valid only for hosts specified in the line.

Example:

```
# turn on idle time monitoring only on host myhost
host "myhost" idle 00:10:00
```

Note: <host> must be included in double quotes.

The **path** condition specifies a file path. If the path exists, then the option is enforced. The syntax of the **path** condition is:

```
path "<path>"   <option>
```

Example:

```
# turn on property "mypapp" if path "/usr/local/myapp" exists
path "myapp" idle "/usr/local/myapp"
```

Note: <path> must be included in double quotes. Use "/" for the directory separator on all platforms, including Windows.

Priority of User Processes

This option specifies the change in the default priority of user jobs, executed by an EnFuzion node. By default, jobs execute under the priority 5 on Windows, which is one level above the screen saver level and under the nice level 10 on Linux/Unix, which is a lower priority than regular processes.

The priority value is system dependent. On Linux/Unix, the nice system call is called with the value supplied. On Windows NT/2000/XP, the value can be between 0 and 15. Values less than 7 lower job priority, and values greater than 7 increase job priority.

The priority of user processes is specified as:

```
prioityoffset <integer>
```

An example for Windows:

```
# Windows NT assigns task priorities as follows:
#  1: System idle process
#  4: Screen saver
#  7: Background user tasks
#  9: Foreground user tasks
# 13: Task Manager
#
# Windows: execute jobs at a low priority, just higher than a screen saver
prioityoffset 5
```

The example changes the default priority of user jobs executed on EnFuzion nodes. If no priority is specified, the value of 5 is used by default on Windows.

An example for Linux/Unix:

```
# Linux/Unix: execute jobs at a low priority, be maximally nice
prioityoffset 10
```

The example sets the nice level of user jobs executed on EnFuzion nodes. If no priority is specified, the value of 10 is used by default on Linux/Unix.

Screen Saver

The **screen saver** option allows EnFuzion jobs to execute only when a screen saver is active or no users other than the EnFuzion user are logged in to the system. If this option is used and the node starts to be used interactively under a user that is different than the EnFuzion user, all EnFuzion jobs on the node are terminated. By default, this option is off.

The screen saver option is specified as:

```
screensaver on | off
```

Example:

```
# Windows: screen saver option:
#     off - available anytime, on - only during an active screen saver
screensaver on
```

This option is implemented only on Windows NT/2000/XP platforms.

Idle Time

Idle time specifies the required lapsed time since the last interactive use of the computer. No EnFuzion jobs are started on the node if the computer is idle for a shorter time period. If this option is used and the node starts to be used interactively, all EnFuzion jobs on the node are terminated. By default, EnFuzion ignores interactive use.

Idle time is specified as:

```
idle <time>
```

Example:

```
# Linux/Unix: available only when not used interactively
idle 00:30:00
```

This option is not implemented on HP Tru64 and Windows NT/2000/XP platforms.

Temporary Disk Space

If the available space in the temporary directory is less than specified, no new EnFuzion jobs are started. The system default value can be changed by the user. The available space is measured in Mb. On Linux/Unix, the temporary directory is **/tmp**. On Windows NT/2000/XP, the temporary directory is defined by system variable %Temp%.

The temporary disk space requirement is specified as:

```
tmpspace <float>
```

Example:

```
# minimum required temporary disk space, in Mb
tmpspace 50
```

Working Disk Space

If the available space in the working directory is less than specified, no new EnFuzion jobs are started. The system default value can be changed by the user. The available space is measured in Mb. On Linux/Unix, the default working directory is the home directory of the user under which the EnFuzion jobs are being executed. On Windows NT/2000/XP, the default working directory is **C:\enfuzion\temp**.

The working disk space requirement is specified as:

```
diskspace <float>
```

Example:

```
# minimum required working disk space, in Mb
diskspace 50
```

Properties

This option specifies all of the properties provided by the node machine. These properties are available globally to all the runs. Properties are user defined.

Properties are specified as:

```
property <property_1>, ..., <property_n>
```

Example:

```
# define user properties
property largemem, printer, app1
```

Used Virtual Memory Space

If the swap and physical memory space used is greater than specified, no new EnFuzion jobs are started. The available space is measured as a percentage of used space compared to the physical memory. For example, 100% means physical memory, 200% means physical memory plus the swap space equal to the physical memory is being used. Total memory used can be seen in the Task Manager item, Commit Charge Total K.

The used virtual memory space is specified as:

```
busyvirtualmemory <float>
```

Example:

```
# Windows: availability upper limit for virtual memory, in % of physical memory
busyvirtualmemory 150
```

This option is implemented only on Windows NT/2000/XP platforms.

Stop Virtual Memory Limit

If the swap and physical memory space used is greater than specified, no new EnFuzion jobs are started, and all currently running processes are killed. The available space is measured as a percentage of used space compared to physical memory. For example, 100% means physical memory, 200% means physical memory plus the swap space equal to the physical memory is being used. Total memory used can be seen in the Task Manager item Commit Charge Total K.

The stop virtual memory limit is specified as:

```
stopvirtualmemory <float>
```

Example:

```
# Windows: virtual memory limit for job termination, in % of physical memory
stopvirtualmemory 200
```

This option is implemented only on Windows NT/2000/XP platforms.

Available Main Memory

If the available main memory is less than specified, no new EnFuzion jobs are started. The system default value can be changed by the user. The available main memory is measured in Kb. On Windows NT, the available main memory is specified by the Performance Monitor Counter "Memory: Available Bytes".

The available main memory requirement is specified as:

```
memory <float>
```

Example:

```
# Windows: minimum required unused physical memory, in Kb
memory 8000
```

This option is implemented only on Windows NT/2000/XP platforms.

Stop Main Memory Limit

If the available main memory is less than specified, no new EnFuzion jobs are started, and all currently running processes are killed.

The system default value can be changed by the user. The available main memory is measured in Kb. On Windows NT/2000/XP, the available main memory is specified by the Performance Monitor Counter "Memory: Available Bytes".

The stop main memory limit is specified as:

```
stopmainmemory <float>
```

Example:

```
# Windows: unused physical memory for job termination, in Kb
stopmainmemory 4000
```

This option is implemented only on Windows NT/2000/XP platforms.

Busy Load Limit

If the load on an EnFuzion node is above this limit, no new jobs are started on the node. On Linux/Unix, the load measured is the first load number returned by the "w" command.

The busy load limit is specified as:

```
busyload <float>
```

Example:

```
# Linux/Unix: availability upper limit for CPU load
busyload 1.00
```

This option is implemented only on Linux/Unix platforms.

Stop Load Limit

If the load on an EnFuzion node is above this limit, existing jobs on the node are stopped. The handling of stopped jobs is specified by the Stop Action option. On Linux/Unix, the load measured is the first load number, returned by the **w** command.

The stop load limit is specified as:

```
stopload <float>
```

Example:

```
# Linux/Unix: CPU load limit for job termination
stopload 3.00
```

This option is implemented only on Linux/Unix platforms.

Busy CPU Usage

If the CPU usage on an EnFuzion node is above this limit, no new jobs are started on the node. On Windows NT/2000/XP, the CPU usage measured is the percentage of CPU time not used by the Idle Thread as specified by the Performance Monitor Counter "System: % Total Processor Time".

The busy CPU usage is specified as:

```
busycpu <integer>
```

Example:

```
# Windows: availability upper limit for CPU usage, in %
busycpu 10
```

This option is implemented only on Windows NT/2000/XP platforms.

Stop CPU Usage

If the CPU usage on an EnFuzion node is above this limit, existing jobs on the node are stopped. The handling of stopped jobs is specified by the Stop Action option. On Windows NT/2000/XP, the CPU usage measured is the percentage of CPU time not used by the Idle Thread as specified by the Performance Monitor Counter "System: % Total Processor Time".

The stop CPU usage is specified as:

```
stopcpu <integer>
```

Example:

```
# Windows: CPU usage limit for job termination, in %
stopcpu 90
```

This option is implemented only on Windows NT/2000/XP platforms.

Busy Processor Queue

If the Processor Queue Length on an EnFuzion node is above this limit, no new jobs are started on the node. On Windows NT/2000/XP, the Processor Queue Length measured is specified by the Performance Monitor Counter "System: Processor Queue Length".

The busy processor queue is specified as:

```
busyqueue <integer>
```

Example:

```
# Windows: availability upper limit for Processor Queue Length
busyqueue 1
```

This option is implemented only on Windows NT/2000/XP platforms.

Stop Processor Queue

If the Processor Queue Length on an EnFuzion node is above this limit, existing jobs on the node are stopped. The handling of stopped jobs is specified by th e Stop Action option. On Windows NT/2000/XP, the Processor Queue Length measured is specified by the Performance Monitor Counter "System: Processor Queue Length".

The stop processor queue is specified as:

```
stopqueue <integer>
```

Example:

```
# Windows: Processor Queue Length limit for job termination
stopqueue 3
```

This option is implemented only on Windows NT/2000/XP platforms.

Off and On Periods

Off periods prevent the execution of EnFuzion jobs during the time specified. On periods overrule the off periods. By default, EnFuzion jobs can run at any time. During off periods, all EnFuzion processes on

the corresponding node are terminated. The processes are started again by the root after the off period expires.

Since node processes are completely terminated during an off period, changes to the **enfuzion.options** file during an off period are not effective until the current off period expires. If an off or an on period is modified in the file during an off period, the modifications go into effect only after the current period expires.

Off periods are specified as:

```
off [ day <day>[-<day>] ] [ time <time>-<time> ]
off date yyyy/mm/dd [ time <time>-<time> ]
```

Examples:

```
# do not execute EnFuzion jobs 7:30-17:30 Mon-Fri
off day Mon-Fri time 7:30-17:30
# do not execute EnFuzion jobs on June 30, 2000
off date 2000/Jun/30
```

On periods are specified as:

```
on [ day <day>[-<day>] ] [ time <time>-<time> ]
on date yyyy/mm/dd [ time <time>-<time> ]
```

Examples:

```
# allow EnFuzion job for 30 minutes at lunch time
on day Mon-Fri time 12:15-12:45
# allow EnFuzion jobs on Jan 1, 2001
on date 2001/1/1
```

Stop Processes

When any of the specified processes are running, no new EnFuzion jobs are started, and all currently running processes are killed. The system default value can be changed by the user. The name of process entered must be the same as that seen in the Task Manager window, without an extension.

Stop processes are specified as:

```
stopproc <process-1>, ..., <process-n>
```

Example:

```
# Windows: host is not available while these processes are executing
stopproc IEXPLORE, cl
```

In the example above, the host is not available for execution of EnFuzion jobs during browsing the Internet or doing some compiling.

This option is implemented only on the Windows NT/2000/XP platforms.

User Busy Condition

With this option, the node calls an external user program to determine if the computer is busy. An optional time value specifies an interval for calling the user program. If no interval is specified, the default value for calling the user program is once a minute.

If the user program returns a non-zero value, then the node is busy and no new jobs are started on the node.

User Busy Condition is specified as:

```
external busy program "<program-path>" [ interval <time> ]
```

Example:

```
# the host is not available if the program returns 1
external busy program "/home/myuser/myprogram" interval 00:01:00
```

The program name **busyload** is reserved for EnFuzion internal use and cannot be used as a user program name.

User Stop Condition

With this option, the node calls an external user program to determine whether the existing jobs on the node should be stopped. An optional time value specifies an interval for calling the user program. If no interval is specified, the default value for calling the user program is once a minute.

If the user program returns a non-zero value, existing jobs on the node are stopped.

User Stop Condition is specified as:

```
external stop program "<program-path>" [ interval <time> ]
```

Example:

```
# jobs are terminated, if the program returns 1
external stop program "/home/myuser/myprogram" interval 00:01:00
```

The program name **stoploa**d is reserved for EnFuzion internal use and cannot be used as a user program name.

Stop Action

This option specifies what happens if an EnFuzion node becomes unavailable during job execution. The job can be either terminated or suspended. If it is terminated, it is automatically rescheduled for later, possibly on some other node. If it is suspended, the EnFuzion node waits for the specified time period. If the node becomes available during that time period, job execution resumes. Otherwise, the job is terminated and rescheduled. The default action is to terminate the job.

The **stopaction** command is specified as one of the following options:

```
stopaction suspend <time>
```

```
stopaction terminate
```

Examples:

```
# Linux/Unix: suspend jobs instead of terminate
stopaction suspend 00:30:00
```

The **stopaction** command with the suspend parameter is not implemented on Windows NT/2000/XP.

Requested Concurrent Jobs

This option specifies the maximum number of concurrent executing jobs on this node.

There is an equivalent configuration option in **node.config** (see the Section called *Requested Concurrent Jobs* for details), which is the recommended use and will be supported in the future. The use of the **joblimit** option in **enfuzion.nodes** is discouraged and might be discontinued in future releases.

The requested concurrent jobs are specified as:

```
joblimit <number>
```

Example:

```
# the number of concurrent jobs
joblimit 1
```

Log File Size

EnFuzion nodes maintain EnFuzion log files in the EnFuzion temporary directory. Default locations are **C:\enfuzion\temp** on Windows and **/tmp** on Linux/Unix. On Windows, the files are named **enfnodea.log** and **enfnodeb.log**. On Linux/Unix, the files are named **.enfnodea.log** and **.enfnodeb.log**.

EnFuzion node processes write their logs to one of the two log files. Two files are used so that records are available at any time. When a file becomes too large, EnFuzion switches the log file.

The size of the files is limited by the loglimit option. The default value for loglimit is 1Mb. The total disk usage by EnFuzion node log files is twice the loglimit size. The loglimit option changes the default log size.

Log file size is specified in Kb as:

```
loglimit <integer>
```

Example:

```
# size of the node log file, in Kb
loglimit 1000
```

Log File Fraction

The logfraction node option specifies how full the current log file must be to trigger the deletion of the other log file.

Log file fraction is specified in % as:

```
logfraction <integer>
```

The default value is 80%.

Example:

```
# log fraction to delete the other log file, in %
logfraction 80
```

Node Directory

This option specifies the EnFuzion node directory. On Linux/Unix, node directories are created under the corresponding user home directories. With this option, a different path for the node directory can be specified. The system default value can be changed by the user.

Node directory is specified as:

```
directory "<location>"
```

Example:

```
# specify the default EnFuzion directory
directory "/tmp/enfnode"
```

Termination Signal

By default, EnFuzion uses the SIGKILL signal to terminate a job. This signal cannot be caught by the job. It unconditionally terminates the job. It is sometimes desirable to terminate the job with a signal that can be caught by the job. With this option, a different termination signal can be specified. The system default option can be changed by the user.

Termination signal is specified as:

```
killsignal <integer>
```

Example:

```
# Linux/Unix: specify the process termination signal
killsignal 3
```

This option has no effect on Windows NT/2000/XP.

Mouse Device

This options specifies the device that is associated with mouse events. This device is monitored by EnFuzion for mouse activity.

On Linux/Unix, the default value is **/dev/mouse**, except on HP-UX where the default value is **/dev/ps2mouse**.

Mouse device is specified as:

```
mouse "<location>"
```

Example:

```
# Linux/Unix: specify mouse device
mouse "/dev/mouse"
```

This option has no effect on Windows NT/2000/XP.

Console Device

This option specifies the terminal device, which is designated as a console. This device is monitored by EnFuzion to determine whether it is executing on the console or not.

On Linux/Unix, the default value is **/dev/console**, except on Linux where the default value is **/dev/tty1**.

Console device is specified as:

```
console "<location>"
```

Example:

```
# Linux/Unix: specify console device
console "/dev/console"
```

This option has no effect on Windows NT/2000/XP.

Sample enfuzion.options File

The following is a sample **enfuzion.options** file. All options are disabled with comments.

To enable load monitoring options, store the text below to file **enfuzion.options** in the EnFuzion directory on the node and modify option values for your environment.

```
#------------ store the text below to enfuzion.options -------------
#
# EnFuzion Node Load Monitoring Options
#       this is only a sample
#       uncomment and modify lines for your configuration

# Windows: execute jobs at a low priority, just higher than a screen saver
#priorityoffset 5
# Linux/Unix: execute jobs at a low priority, be maximally nice
```

```
#priorityoffset 20

# Windows: screen saver option:
#     off - available anytime, on - only during an active screen saver
#screensaver on
# Linux/Unix: available only when not used interactively
#idle 00:30:00

# minimum required temporary disk space in /tmp, in Mb
#tmpspace 50
# minimum required working disk space, in Mb
#diskspace 50

# define user properties
#property largemem, printer, app1

# Windows: availability upper limit for virtual memory, in % of physical memory
#busyvirtualmemory 150
# Windows: virtual memory limit for job termination, in % of physical memory
#stopvirtualmemory 200

# Windows: minimum required unused physical memory, in Kb
#memory 8000
# Windows: unused physical memory for job termination, in Kb
#stopmainmemory 4000

# Linux/Unix: availability upper limit for CPU load
#busyload 1.00
# Linux/Unix: CPU load limit for job termination
#stopload 3.00

# Windows: availability upper limit for CPU usage, in %
#busycpu 10
# Windows: CPU usage limit for job termination, in %
#stopcpu 90

# Windows: availability upper limit for Processor Queue Length
#busyqueue 1
# Windows: Processor Queue Length limit for job termination
#stopqueue 3

# do not execute EnFuzion jobs 7:30-17:30 Mon-Fri
#off day Mon-Fri time 7:30-17:30
# do not execute EnFuzion jobs on June 30, 2000
#off date 2000/Jun/30
# allow EnFuzion jobs for 30 minutes at lunch time
#on day Mon-Fri time 12:15-12:45
# allow EnFuzion jobs on Jan 1, 2001
#on date 2001/1/1

# Windows: host is not available while these processes are executing
#stopproc IEXPLORE, cl
```

```
# the host is not available if the program returns 1
#external busy program "/home/myuser/myprogram" interval 00:01:00
# jobs are terminated, if the program returns 1
#external stop program "/home/myuser/myprogram" interval 00:01:00

# Linux/Unix: suspend jobs instead of terminate
#stopaction suspend 00:30:00

# the number of concurrent jobs
#joblimit 1

# size of the node log file, in Kb
#loglimit 1000
# log fraction to delete the other log file, in %
#logfraction 80

# specify the default EnFuzion directory
#directory "/tmp/enfnode"
# Linux/Unix: specify the process termination signal
#killsignal 3
# Linux/Unix: specify mouse device
#mouse "/dev/mouse"
# Linux/Unix: specify console device
#console "/dev/console"

#-------------------------------------------------------------------
```

Specifying Environment Variables

The **environment** configuration file sets the values of environment variables for user programs that are executed by EnFuzion.

The **environment** file is read when the node server is started. If any of the environment values are changed, then the node server must be terminated and restarted for any changes to take the effect.

The rest of this section provides details about the **environment** file.

The environment File

EnFuzion checks for the **environment** file in the following locations: the local working directory, the directory specified in the ENFNODE_PATH environment variable, and the main EnFuzion installation directory on the node. By default, the **environment** file is located in the main EnFuzion installation directory on the node.

On Linux/Unix, the default location of the main EnFuzion installation directory is **$HOME/enfuzion**.

On Windows, the default location of the main EnFuzion installation directory is **C:\enfuzion**.

The **environment** file contains lines which describe environment variables. Variables can be either assigned a new value or modified. Lines that start with "#" are treated as comments and ignored.

A new variable value can be defined or the value of an existing variable can be modified with an assignment:

```
<name>=<value>
```

An example:

```
HOSTADDR=testhost
```

A new variable value can be defined or a string can be added to an existing variable with a concatenation:

```
<name>+=<value>
```

An example:

```
PATH+=:/usr/local/appdir
```

This form is especially useful for the PATH environment variable.

Specifying Path Correspondence

The **paths** file allows you to specify how file paths on EnFuzion nodes correspond to paths on submit computers. This is useful for heterogeneous configurations where compute nodes might use a different operating system from the submit computer. At the job execution time, EnFuzion changes file references from the submit computer to the local node, according to instructions in the **paths** file.

The **paths** file is read when a job is started on the node, so there is no need to restart the node server, when the file is changed.

The **paths** file works in conjunction with two run options, ENFPATH_SUBSTITUTE and ENFSUBMIT_PLATFORM. ENFPATH_SUBSTITUTE is a list of run variables, which EnFuzion will change at the job execution time. ENFSUBMIT_PLATFORM provides the operating system on the submit computer. It can be one of the following values: "windows", "linux" or "osx".

ENFPATH_SUBSTITUTE and ENFSUBMIT_PLATFORM can be set in the run file. Examples:

```
# the submit computer runs Windows
set ENFSUBMIT_PLATFORM "windows";

# substitute variables "inputpath" and "outputdir"
set ENFPATH_SUBSTITUTE "inputpath,outputdir";
```

The rest of this section provides details about the **paths** file.

The paths File

EnFuzion checks for the **paths** file in the following locations: the local working directory, the directory specified in the ENFNODE_PATH environment variable, and the main EnFuzion installation directory on the node. By default, the **paths** file is located in the main EnFuzion installation directory on the node.

On Linux/Unix, the default location of the main EnFuzion installation directory is **$HOME/enfuzion**.

On Windows, the default location of the main EnFuzion installation directory is **C:\enfuzion**.

Each line in the **paths** file contains a correspondence for one file path. There is no limit on the number of lines, so multiple file paths can be specified. If the first character is '#', the line is a comment.

Each line consists of a list of keywords for the operating system, each keyword followed by a path on that operating system. Use "/" for the directory separator in the path on all operating systems, including Windows. Keywords are:

- **windows**, path on Windows submit clients;
- **osx**, path on Mac OS X submit clients;
- **linux**, path on Linux submit clients;
- **windowsnode**, path on Windows compute nodes;
- **osxnode**, path on Mac OS X compute nodes;
- **linuxnode**, path on Linux compute nodes.

Examples:

```
# Windows drive F: corresponds to
# /mnt/share on Mac OS X and Linux
windows "F:" osxnode "/mnt/share" linux "/mnt/share"

# Windows directory //myhost/repository/ corresponds to
# /private/var/repository on Mac OS X
windows "//myhost/repository/" osxnode "/private/var/repository"

# Mac OS X directory "/Users/john/repository" corresponds to
# //mac/repository on Windows and
# /mnt/repository on Linux
osx "/Users/john/repository" windowsnode "//mac/repository" linuxnode "/mnt/repository"
```

Specifying Startup Script

EnFuzion allows users to supply a node startup script for each Windows based EnFuzion node. The script is called **startup.bat**. EnFuzion executes this startup script each time a node is started. The startup script can be used to perform any user defined actions when the node starts, for example, mapping a shared file repository to a local drive letter.

The rest of this section provides details about the **startup.bat** script.

The startup.bat Script

The **startup.bat** script must be placed in the **bin** subdirectory of the EnFuzion installation directory. On Windows, the default location of the main EnFuzion installation directory is **C:\enfuzion**.

The script must be a Windows batch file. There is no equivalent to **startup.bat** on Mac OS X and Linux platforms.

Node Based Security Features

Node based security features include trusted hosts and executables, password encryption over the network and root authentication.

Trusted Hosts and Executables

Trusted hosts and trusted executables provide a straightforward method to limit hosts that can access EnFuzion nodes and user programs that EnFuzion is able to run. Only trusted hosts are allowed to access an EnFuzion node. Only trusted executables are executed by EnFuzion on the node.

The enfuzion.security File

Trusted hosts and trusted executables are specified in the **enfuzion.security** file. A system security file and a user specific security file can be located on each node host.

On Linux/Unix nodes, the system security file must be located in the directory **/var/opt/enfuzion** and the user security file in the directory **~/enfuzion/enfuzion.security**.

On Windows NT nodes, the system security file must be located in the directory **C:\enfuzion\enfuzion.security** (your drive letter may be different). User security files are not supported on Windows NT.

Security files are handled as follows:

- If the system security file exists, then the system security file is used and the user security file is ignored.
- If the system security file is not found and the user security file exists, then the user security file is used.
- If no security files are found, then all hosts and executables are trusted and no limitations are imposed on either hosts or executables.

If a security file exists, only hosts and executables specified in the file are trusted. An empty **enfuzion.security** file therefore specifies that no hosts and no executables are trusted.

Note: There are no special provisions for the local host address or for the 127.0.0.1 address. If the **enfuzion.security** file is enabled and access from the local host is required, then these addresses must be explicitly allowed.

File Syntax

Security files are text files containing a list of security specifications. Each line represents one specification. Comment lines begin with the ’pound mark’, "#".

The syntax of a security specification is:

```
<security_status>  <resource_type>  <resource_name_list>
```

- **<security_status>**

  It can be either allow or deny. Allow specifies a list of trusted resources. Deny specifies a list of resources that are not trusted.

  For example, the following lines specify EnFuzion root hosts pluto and mini as trusted and host garfield as not trusted:

  ```
  allow host pluto, mini
  deny  host garfield
  ```

- **<resource_type>**

  It provides the type of resource. <resource_type> can be either **host** or **executable**.

  For example, the following line specifies executables echo and ps as trusted:

  ```
  allow executable echo, ps
  ```

- **<resource_name_list>**

  It gives a list of resource names. The names are separated by ’,’. The ’*’ denotes all resources of the given type. For example, the following line specifies all executables as trusted:

  ```
  allow executable *
  ```

If the <resource_type> is host, the names of hosts in a list can be either Internet IP addresses or DNS host names. Internet IP addresses have the format d.d.d.d[/m], where d is a decimal number or the wild card character ’*’. The character ’*’ stands for any number. Parameter m can be used to specify network addresses. It determines the number of bits in the IP address that are used for address matching. A DNS host name consists of a host name and an optional domain name. The ’*’ as a host name denotes all hosts. For example, the following line specifies all hosts on the network 192.166.2 as trusted:

```
allow host 192.166.2.*
```

To specify the same allowed host with network addressing:

```
allow host 192.166.2.0/24
```

> **Note:** The use of IP addresses is strongly recommended. If DNS host names are used instead, they can cause significant delays in EnFuzion operation. Due to DNS resolution, it can take up to several minutes to resolve an official host name on some networks.

The order of security specifications in a security file is important. All the specifications in the security file are verified for a given resource and the last status is taken as valid. For example, the following lines specify host strippy as trusted:

```
deny host strippy
allow host strippy
```

The following lines specify host strippy as not trusted:

```
allow host strippy
deny host strippy
```

The **enfuzion.security** file containing the following lines specifies all hosts running any executables as trusted:

```
allow host *
allow executable *
```

This is equivalent to not using a security file.

Security Considerations in Job Execution Commands

When trusted executables are used, note that all command paths must be fixed at the time of execution. For example, shell constructs that are interpreted after a command is submitted for execution are not allowed:

```
execute $HOME/bin/enfecho
```

In the example above the complete path of the command is not determined until the line is executed by the shell. Therefore the security status of the command cannot be predetermined and the command is rejected. The command must be rewritten as follows:

```
execute /home/myuser/bin/enfecho
```

User Defined Decryption Primitives

EnFuzion supports user defined decryption primitives. These primitives are implemented by using a dynamic library. The library contains user specific decryption methods. If the library exists, the primitives from the library are used instead of the default EnFuzion primitives.

This feature is supported only on Windows NT/2000/XP.

Overview of the Dynamic Library

The dynamic library for user defined decryption supports two tasks, the decryption of user passwords and the decryption of the EnFuzion security file, **enfuzion.security**. The library provides an interface which specifies decryption functions called by EnFuzion.

The library must have the filename **enfuser.dll**. The **.dll** file which is usually in the directory **C:\enfuzion\bin** must be in the search path of each EnFuzion node.

The library is loaded at program startup. If a new version of the library is provided by the user, all EnFuzion programs that use the library must be restarted. The programs affected are Starter Service and node server.

Interface

The library supports decryption of passwords and decryption of the file **enfuzion.security**.

The following interface functions are defined for **enfuser.dll**:

```
int decryptPassword(
     char *passin,
     int inlen,
     char *passout,
     int outlen);

void *openFileDecryption(
     char *filename);

int readNextDecryptedLine(
     void *fid,
     char *buffer,
     int buflen);

int closeFileDecryption(
     void *fid);
```

If a function is not found in **enfuser.dll**, then its default version is called.

A binary dump of **enfuser.dll** is similar to the following output:

```
d:\enfuzion\bin> dumpbin -exports enfuser.dll

Dump of file enfuser.dll
...
File Type: DLL

       Section contains the following Exports for secdemodll.dll
...
        ordinal hint   name

              5    0   __DebuggerHookData  (000021BC)
```

```
3    1   _closeFileDecryption  (000014BE)
4    2   _decryptPassword  (000014D7)
1    3   _openFileDecryption  (00001480)
2    4   _readNextDecryptedLine  (00001495)
```

Notice that the names of library functions are preceded by an underscore, "_".

Because **enfuser.dll** is used also by processes running in the background, functions should not write to standard output or standard error streams.

Details of the interface functions are described in the next section.

Decryption of Passwords

Passwords to access remote machines are specified in the network configuration file, usually **enfuzion.nodes**. These passwords can be in a clear text form or encoded by EnFuzion, as described in the Section called *Encrypted Passwords in enfuzion.nodes* in Chapter 6.

The user can replace clear text passwords in **enfuzion.nodes** with encrypted passwords. Encrypted passwords must consist of non-white, printable ASCII characters. These passwords will be decrypted by calling function decryptPassword() from the dynamic library **enfuser.dll**. If **enfuser.dll** is not found or decryptPassword() is not defined, then no user decryption will be performed.

Function decryptPassword() provides the following interface:

```
int decryptPassword(char *passin, int inlen, char *passout, int outlen);
```

The passin pointer points to the user encrypted password from **enfuzion.nodes**. The inlen parameter is the number of valid characters in passin. The passout pointer points to the decrypted password. The decrypted password must contain non-white, printable ASCII characters, terminated by a null character, ’\0’. This string will be used as a password to perform a login on the remote host. The outlen parameter is the length of the passout buffer, available for the decrypted password. The function implementation must handle passout buffer overflows. The function returns 0 on success or a negative user defined error code otherwise.

Currently, EnFuzion provides a passout buffer size of 1024 characters, i.e., outlen equals 1024.

User encrypted passwords can be decrypted by EnFuzion, but not vice-versa.

Decryption of enfuzion.security

On Windows NT/2000/XP, the EnFuzion security file **enfuzion.security** can be encrypted by the user. EnFuzion uses the following functions from **enfuser.dll** to decrypt the file.

```
void *openFileDecryption(char *filename);
```

This function returns a handle to identify an encrypted file. NULL is returned on error. Currently, EnFuzion calls openFileDecryption() with the filename parameter containing the absolute path of the **enfuzion.security** file on the host.

```
int readNextDecryptedLine(void *fid, char *buffer, int buflen);
```

This function reads the next line from the file and returns the number of bytes read or a negative number if error. A return value of 0 indicates an end of file. Currently, EnFuzion provides a buffer of size 1024 characters, i.e., buflen equals 1024. fid is equal to the handle, returned by a previous call to openFileDecryption().

```
int closeFileDecryption(void *fid);
```

Closes the handle. Returns 0 on success and -1 otherwise. fid equals to the handle, returned by a previous call to openFileDecryption().

If a function is not defined in **enfuser.dll**, then its default version with no decryption is called.

Library Template

This example provides a library template of function implementations. The template provides no decryption and handles clear text.

```
/*
 * Sample implementation of security dll: "enfuser.dll"
 */

#include <stdio.h>
#include <string.h>

/*
 * Decrypt password
 * return 0 on success
 *       -1 on error
 */
int _export decryptPassword(char *password, int passlen,
                        char *decryptedpassword, int decpasslen)
{
  if (strlen(password) > decpasslen) {
    return -1;
  }
  strcpy(decryptedpassword, password);
  return 0;
}

/*
 * Return a handle to decrypted file
 * or NULL on error.
 */
void * _export openFileDecryption(char *filename)
{
  return(fopen(filename, "r"));
}

/*
 * Read next decrypted line
 * Return number of characters read
 */
```

```
int _export readNextDecryptedLine(void *fid, char *outbuf, int outbuflen)
{
   if (fgets(outbuf, outbuflen, fid) == NULL) {
     return 0;
   }
   return strlen(outbuf);
}

/*
 * Close handle to decrypted file
 * Return 0 on success
 *        -1 on error
 */
int _export closeFileDecryption(void *fid)
{
  if (fclose(fid) != 0) {
    return -1;
  }
  return 0;
}
```

Root Authentication

EnFuzion root authentication is based on public/private key encryption. This authentication strengthens network security on nodes, since it assures that only authorized root hosts are able to use remote nodes. Root authentication is supported only on Windows NT/2000/XP nodes.

Root authentication works as follows. The root host distributes its public keys to the nodes. When public keys are present on a node, the node initiates the authentication of the root's identity. If the root host does not have the matching private key, authentication fails and the connection to the root host is terminated.

Private and public keys can be generated and distributed with the EnFuzion provided Enfkey utility, described below.

EnFuzion provides a library to implement the root authentication capability. This authentication library can be replaced with a user defined library, which might be required in certain high security environments.

The following sections describe the Enfkey utility, the process of generating and distributing the keys, the EnFuzion provided authentication library and how to implement a user defined authentication library.

The Enfkey Utility

The Enfkey utility generates a public/private key pair. If a user defined authentication library is provided, enfkey uses that library.

The Enfkey program uses the following syntax:

```
enfkey keygen
```

This generates new public and private keys for the system where enfkey is executed. The IP address of the system that generated the keys is also printed to the standard output.

Example:

For the default EnFuzion authentication library, the keys are placed in file **enf_key.priv**. A sample file contents is:

```
Id=172.12.85.23
PrivKey=11773C2ADB11EBE6FBE7911056C3A1E53A4C7F4B
PublicKey=97F035A7B89B95CBA91F3EE1E3343293CACDDECD59D7
CA381490532BB118ECD204703702137E80CFB89EA622CE153699DE
2060CDB787A153B6321CFC376C7C97913D3C1795015A10FC3C9935
236DD68C2C3BC11E9142787600361F1AEF9EC9B82137270E1F175A
A1F52836030776AE0DA6FE5E4CB5E1C16C0EC60058DC0F47F1
```

Id designates the IP address of the system where the keys were created. PrivKey and PublicKey contain private and public keys, respectively.

Generation and Installation of Keys

The procedure to generate the keys on the root system and store the public key on the node is described below.

- On the root, generate its public/private pair of keys.

  ```
  enfkey keygen
  ```

- On the root, make a duplicate of the **enf_key.priv** file and remove the PrivKey field with the private key.
- Copy the duplicate **enf_key.priv** file with the private key removed to the node.
- On the node, install the EnFuzion root public key by adding the contents of the duplicate **enf_key.priv** file to the enfuzion.key.

EnFuzion Provided Authentication Library

The EnFuzion node package includes an authentication library, which is based on the widely used standard OpenSSL library. A corresponding SSL library must be available on the EnFuzion root system. The EnFuzion Windows package already includes all required libraries for the root and the node systems. Most Linux/Unix systems have the OpenSSL library already installed. Otherwise, the OpenSSL library must be installed by the user. For environments with specific authentication requirements, the authentication library can be replaced with a user provided library (see the Section called *User Defined Authentication Primitives* for more information).

The EnFuzion root authentication is performed, if at least one public key is installed. Otherwise, any EnFuzion root system can access the node.

The default authentication library stores public keys on nodes in file **enfuzion.key**. Only Windows NT/2000/XP platforms are supported. The file is located in the EnFuzion directory, which is **C:\enfuzion** by default.

Private keys on EnFuzion root systems are stored in file **enfuzion.pkey**. Any EnFuzion supported platform can act as the authenticated root. File **enfuzion.pkey** is located in the EnFuzion **config** directory.

The user can manually add public keys to the **enfuzion.key** file on the node. To add a public key, make a duplicate of **enf_key.priv** file on the root, delete the private key line in the duplicate, copy the modified duplicate file to the node and append the file to the **enfuzion.key** file on the node. The **enfuzion.key** file can contain multiple public keys.

User Defined Authentication Primitives

Default EnFuzion provided authentication primitives can changed by replacing the default authentication library provided with EnFuzion with a custom library. The custom library must be available both on the root host and on all node hosts. This section describes the interface used by the authentication library.

Overview of the Dynamic Library

The dynamic library provides primitives for root authentication called by EnFuzion. It supports the following tasks: generation of private/public keys, and adding and removing keys to a machine configuration. The library must have the filename **enfauth.dll** on Windows or **enfauth.so** on Linux/Unix hosts. The dynamic library file, which is usually in the directory **/enfuzion/bin** on Windows NT/2000/XP, must be in the search path of each EnFuzion node.

Interface

The following interface functions are defined for the library:

```
int CryptoCapabilities();

char *CryptoInformation();

int CryptoSignBuffer(
     char  *fromIP,
     char  *buff,
     int    len,
     char **dest);

int CryptoVerifyBuffer(
     char  *fromIP,
     char  *originalBuff,
     int    len,
     char  *signatureBuffer);

int CryptoGenKeys(
     char **PublicKey,
     char **PrivateKey);

int CryptoAddKey(
```

```
        char  *fromIP,
        char  *PublicKey,
        char  *PrivateKey);

    int CryptoRemoveKey(
        char  *forIP);
```

If a function is not found in **enfauth.***, then its default version is used. A binary dump of **enfauth.dll** on Windows produces the following output:

```
d:\enfuzion\bin> dumpbin enfauth.dll /EXPORTS

Dump of file enfauth.dll
...
File Type: DLL

...

ordinal hint RVA      name

      1    0 00001023 CryptoAddKey
      2    1 00001019 CryptoCapabilities
      3    2 0000100A CryptoGenKeys
      4    3 00001014 CryptoRemoveKey
      5    4 00001028 CryptoSignBuffer
      6    5 0000100F CryptoVerifyBuffer
      7    5 0000102D CryptoInformation
```

Windows note:

The function name has no decorations. The function must be implemented with the STDCALL calling convention. The library must be compiled as a multi threaded DLL.

Linux/Unix note:

Because **enfauth.dll/so** is used by processes running in the background, functions must not write to standard output or standard error streams.

Returning Status

Each library function must return status. If the function is successful it must return 0. In the case of an error, the function must return one of the following error codes.

List of valid error codes:

```
#define AUTH_ERROR_NO_CAPABILITIES -1
#define AUTH_ERROR_NOT_TRUSTED     -3
#define AUTH_ERROR_NO_NETWORK      -4
#define AUTH_ERROR_CANNOT_READ     -5
#define AUTH_ERROR_CANNOT_SEND     -6
#define AUTH_ERROR_NO_PRIVATEKEY   -11
#define AUTH_ERROR_NO_PUBLICKEY    -12
```

Defining Library Capabilities

```
int CryptoCapabilities();
```

The function returns an integer defining which functions from the library can be used.

The following flags are valid:

```
#define AUTH_CAP_SIGN       1
#define AUTH_CAP_VERIFY     2
#define AUTH_CAP_ADDKEY     4
#define AUTH_CAP_REMOVEKEY  8
#define AUTH_CAP_GENKEY     16
```

Displaying Library Information

```
char *CryptoInformation();
```

The function returns a static string with information about the library. The information is recorded in EnFuzion logs.

Signing Buffer

```
int CryptoSignBuffer(
     char  *fromIP,
     char  *buff,
     int    len,
     char **dest);
```

During the authentication sequence, the root host is requested to encode random data. EnFuzion calls CryptoSignBuffer to perform this task.

The parameter fromIP holds an IP number as text (e.g. "172.20.93.19") defining who requested the signature. Random data and the length of the buffer are held in the next two parameters. This function stores encoded data in a text string dest.

Verifying Returned Buffer

```
int CryptoVerifyBuffer(
     char  *fromIP,
     char  *buff,
     int    len,
     char  *signatureBuffer);
```

A node that authenticates the root host needs to verify the returned result. EnFuzion calls CryptoVerifyBuffer to perform this task. Input parameter fromIP defines the root IP number as text (e.g. "172.20.93.18"). Parameters buff and len define data. The last parameter is the text string received from the root.

Adding Keys to a Node or a Root Host

```
int CryptoAddKey(
     char  *assignedIP,
     char  *PublicKey,
     char  *PrivateKey);
```

In order to perform authentication, the root must store its private and public keys and the node host must store the root's public key. EnFuzion calls the function CryptoAddKey on every node with the input parameters PublicKey and PrivateKey set to NULL. If CryptoAddKey is called on the root machine PublicKey and PrivateKey are not NULL. Parameter assignedIP points to the IP as text (e.g. "172.20.93.18") identifying which computer owns this public key. The function is also called on the root host if assignedIP points to the local host. This time the private key is also passed in the parameters.

Removing Keys from a Node

```
int CryptoRemoveKey(char *forIP);
```

The function CryptoRemoveKey removes a key. Sometimes keys have to be replaced on a particular host. In order to retain access to that host, the user must first install the new keys and then remove the old ones. Parameter forIP points to the text IP (e.g. "172.20.93.18") of the owner of the public key that the user wants to remove.

Generating New Keys

```
int CryptoGenKeys(char **PublicKey,
     char **PrivateKey);
```

Newly generated keys can be installed on the root and on node hosts. The function CryptoGenKeys must allocate space and copy the private and public keys to that space. These keys must be in text form. EnFuzion releases buffer space when they are no longer needed.

Library Template

The following example is a library template. The template provides only a framework, but no *real* encryption or decryption .

Example:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>


#ifdef WIN32
#include <windows.h>
#define CALL_CONV __stdcall
#else
```

```
#include <sys/stat.h>
#include <sys/types.h>
#include <netdb.h>
#include <unistd.h>
#define CALL_CONV
#endif

static char *lib_info_text = "enFuzion authentication library template";

char* CALL_CONV CryptoInformation()
{
  return lib_info_text;
}

int CALL_CONV CryptoCapabilities()
{
  return 0x01 | 0x02 | 0x04 | 0x08 | 0x10;
}

int signBuff(char  *buff,
             int    len,
             char **dest,
             int    key)
{
  return 0;
}

int getKey(char *ipBuff, int *keynum)
{
  FILE *file;
  char iptmp[256];

  file = fopen("/enftmp.key", "r");
  if (file) {
    while (!feof(file)) {
      fscanf(file, "%s %d\n", iptmp, keynum);
      if (strcmp(iptmp, ipBuff) == 0) {
        fclose(file);
        return 0;
      }
    }
    fclose(file);
  }
  return -1;
}

int CALL_CONV CryptoSignBuffer(char  *fromIP,
                     char  *buff,
                     int    len,
                     char **dest)
{
  struct hostent *hostent ;
  int            ret, i;
```

```
  char           tmpBuff[256];
  unsigned char  sig[5];

  gethostname(tmpBuff, 256);
  hostent = gethostbyname(tmpBuff);
  for(i = 0; i < 4; i++)  sig[i] = hostent->h_addr[i];
    sprintf(tmpBuff, "%u.%u.%u.%u", sig[0], sig[1], sig[2], sig[3]);

  /* write private file for license issuer */

  ret = getKey(tmpBuff, &i);
  if (ret) {
    /* return error - no private key */
    return -11;
  }

  /* no encryption - return private key since public key is same */
  sprintf(tmpBuff, "%d", i);
  *dest = strdup(tmpBuff);

  return 0;
}

int CALL_CONV CryptoVerifyBuffer(
                      char  *fromIP,
                      char  *originalBuff,
                      int    len,
                      char  *signatureBuffer)
{
  int           ret, i;
  char          tmpBuff[256];

  ret = getKey(fromIP, &i);
  if (ret) {
    /* return error - no public key */
    return -12;
  }

  sprintf(tmpBuff, "%d", i);

  /* verify the "signature" */
  if (strcmp(signatureBuffer, tmpBuff) == 0)
  {
    return 0;
  }
  /* return error - not trusted */
  return -3;
}

int CALL_CONV CryptoGenKeys(char **PublicKey, char **PrivateKey)
{
  struct hostent *hostent ;
  int     i;
```

```
  char      temp[256];
  unsigned char sig[5];

  gethostname(temp, 256);
  hostent = gethostbyname(temp);
  if (hostent)
  {
   /* write private file for license issuer */
    for(i = 0; i < 4; i++)  sig[i] = hostent->h_addr[i];
      sprintf(temp, "%u.%u.%u.%u", sig[0], sig[1], sig[2], sig[3]);

    srand(time(0));
    i = rand();
    sprintf(temp, "%d", i);

    *PublicKey = strdup(temp);
    *PrivateKey = strdup(temp);

    return 0;
  }

  /* return error - no capabilities */
  return -1;

}

/* install public key or private key */
int CALL_CONV CryptoAddKey(char *forIP, char *PublicKey, char *PrivateKey)
{
  FILE *file;

  if (PrivateKey != 0 &&
      PublicKey  != 0)
  {
    /* final stage - we are on root */
    file = fopen("/enftmp.key", "a");
    if (file) {
      fprintf(file,"%s %s\n", forIP, PublicKey);
      fclose(file);
    }
  } else if (PublicKey != 0)
  {
    /* install only public keys */
    file = fopen("/enftmp.key", "a");
    if (file) {
      fprintf(file,"%s %s\n", forIP, PublicKey);
      fclose(file);
    }
  }
  return 0;
}

int CALL_CONV CryptoRemoveKey(char *forIP) /* remove key */
```

```
{
  char buff[256];
  fpos_t pos;
  int   i;
  FILE  *file;

  file = fopen("/enftmp.key", "r+");
  while (file &&
        !feof(file))
  {
    fgetpos(file, &pos);

    if  (fscanf(file,"%s %d\n", buff, &i) > 0 &&
         strcmp(buff, forIP) == 0)
    {
      buff[0] = '#';
      fsetpos(file, &pos);
      fwrite(buff, 1, 1, file);
      break;
    }
  }
  if (file)
    fclose(file);
  return 0;
}
```

Chapter 8. Run Description

Introduction

A run is a description of jobs to execute. Each run can contain one or more jobs. A run can be either a command line program, a shell script or a parametric execution, containing many jobs. A parametric execution consists of jobs that execute the same application with different input parameters. A run describing a parametric execution contains a list of commands to execute, a list of input values for each job and any additional configuration options.

EnFuzion provides two different ways to describe a parametric execution, either as a plan file, or as a run file. Plan files and run files are similar. The major difference is that run files contain specific values for input parameters, while plan files contain only descriptions of job parameters, but not their actual values. A plan file is a template for the run. Plan files are used by EnFuzion to build application specific GUIs, allowing users to quickly generate jobs for parametric executions. A plan file must be converted to a run file, before it can be submitted to the EnFuzion root for execution.

Plan files are regular text files. EnFuzion includes the Preparator program, which provides a simple method for creating plan files. Alternatively, plan files can be created with standard text editors. Plan files are converted to run files with the EnFuzion Generator program. Using the plan file, the Generator creates an application specific GUI, which is used to select values for job parameters and produce a run file. Run files are complete run descriptions. They can be submitted directly to the EnFuzion root for execution. A run file includes a description of the run and values for the job parameters.

Run files are regular text files. Depending on the application, they can be produced by using the Preparator and the Generator, by using standard text editors, or can be generated by other programs.

Plan and run files are usually created on a submit computer, which can be a workstation or a personal computer. The Preparator and the Generator are also executed on the submit computer.

EnFuzion provides additional capabilities for handling the demands of the most complex environments. These capabilities include support for concurrent execution of multiple runs with priorities, resource management, datajobs, and handling timeouts and errors.

The sections below describe runs as a command line, a script or a parametric execution, including the creation of plan files, the creation of run files and a detailed description of plan and run files. Descriptions are provided also for multiple runs, resource management, datajobs and handling of timeouts and errors.

Command Line Programs

When a run is submitted as a command line program, there is no need to prepare any special files. The program and its options are simply provided on the command line.

A command line program is submitted as follows:

```
enfsub  [ <enfsub_options> ]  <program>  [ <program_options> ]
```

The user program and its options are provided as parameters to the **enfsub** program. They can be preceded by <**enfsub_options**>, which are **enfsub** specific parameters.

An example:

```
enfsub sleep 30
```

Details about **enfsub** and its options are provided in the Section called *The Enfsub Program* in Chapter 10.

The following is a more complex command line example for Windows:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cmd /c copy input.txt output.file"
```

The following is the same command line example for Linux/Unix:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-\$ENFJOBNAME-\$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cp input.txt output.file"
```

Scripts

When a run is submitted as a script, the script and its options are simply provided on the command line. Additional **enfsub** options can be specified in the script to avoid the need to place them on the command line every time the script is executed.

Scripts are submitted similarly to command line programs:

```
enfsub  [ <enfsub_options> ]  <script>  [ <script_options> ]
```

The script and its options are provided as parameters to the **enfsub** program. They can be preceded by **<enfsub_options>**, which are **enfsub** specific parameters. Details about **enfsub** and its options are provided in the Section called *The Enfsub Program* in Chapter 10. The **<script>** file must be already available on the node or copied explicitly as part of the **enfsub** command.

<enfsub_options> can also be specified inside a script. The **enfsub** program identifies scripts as follows. On Linux/Unix, scripts start with a string "#!". On Windows, scripts are files that have the suffix **.bat**. If **enfsub** detects a script as opposed to a binary executable, it checks the script for options. On Linux/Unix, options are provided in lines that start with "#ENF". On Windows, options are provided in lines that start with "@rem ENF" or "rem ENF". If an option is specified on the command line and in the script, then the command line value takes precedence.

An example:

```
enfsub -i myscript.sh myscript.sh
```

This example copies the script **myscript.sh** to the node and executes it.

The following is a Windows script that has the same effect as the command line example in the previous section:

```
@echo off
rem ENF -i script.bat
rem ENF -n sample -a myaccount
rem ENF -i input.txt
rem ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
rem ENF -rd -count 2 -e user@domain.com -m d

copy input.txt output.file
```

The script is submitted with:

```
enfsub script.bat
```

The following is a Linux/Unix script that has the same effect as the command line example above:

```
#!/bin/sh
#ENF -i script.sh
#ENF -n sample -a myaccount
#ENF -i input.txt
#ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
#ENF -rd -count 2 -e user@domain.com -m d

cp input.txt output.file
```

The script is submitted with:

```
enfsub ./script.sh
```

Parametric Executions

When a run represents a parametric execution with many jobs, the run needs to be described in a plan file or a run file. This section gives details about plan and run files.

A plan file is a template for the run. It includes commands to be executed for each job in the run and descriptions of job parameters, but does not include actual input values for job parameters.

A run file is used to submit the jobs for execution. It contains job commands and jobs with their corresponding parameter input values.

On Windows, runs can be submitted with a double click on the run file. On Linux/Unix or from the command line, runs are submitted for execution with the **enfsub** program:

```
enfsub  [ <enfsub_options> ]  [ -run ]  <run_file>  [ <input_files> ]
```

<run_file> and its **<input_files>** are submitted for execution to the Dispatcher.

The **-run** option can be omitted, if the **<run_file>** ends with the **.run** suffix. Additionally, the **enfsub** program automatically detects input files, so the **<input_files>** arguments can be omitted from the command line as well.

The following sections describe plan and run files in detail.

Creating a Plan File

Plan files are regular text files. The files can be created and modified with any standard text editor. EnFuzion provides a simple tool, the Preparator, to assist in creating plan files. The Preparator provides a simple text editor as well as a wizard, which covers major stages of plan creation. The rest of this section provides details on how to create plans with the Preparator.

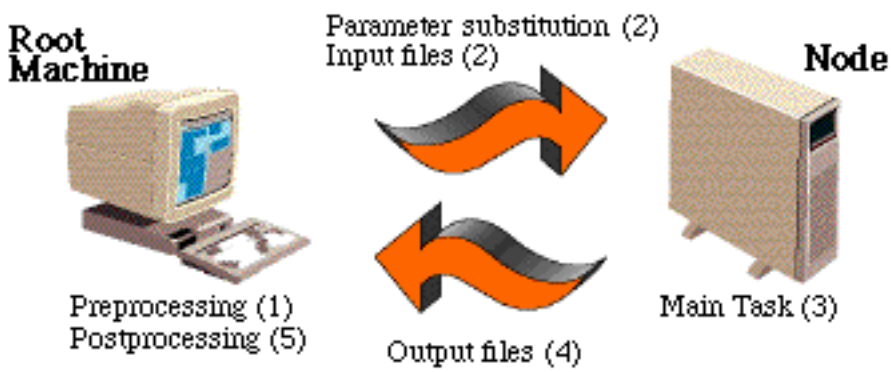
The Preparator

The Preparator allows you to build a plan without explicitly writing any EnFuzion commands. It is designed to allow easy creation of plans for the most common uses of EnFuzion. Specifically, it supports the following five phases of creating a plan for distributed execution of jobs:

- Execution of preprocessing commands on the root computer before any jobs are started
- Copying the necessary input files to remote computers and the replacement of parameter placeholders with input values for the job
- Execution of the user commands on remote computers
- Copying the output files back to the root computer
- Execution of post processing commands on the root computer at the end of execution

These phases are summarized in Figure 8-1.

Figure 8-1. Phases of Standard EnFuzion Computation



The Preparator can be started on a command line as:

```
enfpreparator [ <plan_name> ]
```

The main window of the Preparator is a simple text editor, which allows you to modify the plan file at any time. A new plan can be easily created with a Preparator wizard, which guides you through the process of plan creation.

When you start the Preparator with an existing plan, the plan text will be displayed in the main window. You can use this window to edit the plan. When you start the Preparator without a plan, you will automatically be presented with the wizard. The wizard can be started at any time through the **File** menu the **Wizard** option or with the **Wizard** button in the toolbar.

A plan can be saved through the **File** menu, options **Save** and **Save As ...**.

The **File** menu, option **Generator**, or the **Generator** button in the toolbar save the plan and submit the plan to the EnFuzion Generator, which is described in a later section (see the Section called *The Generator*).

The following sections describe the Preparator wizard in more detail.

Preparator Wizard

The wizard provides the following dialogs for building the plan:

- Introduction
- Parameter Description
- Preprocessing
- Input files
- Substitution Files
- User Commands
- Output files
- Post processing
- Finishing Dialog

Introduction

This dialog explains a few simple facts about the wizard. It is possible to cancel the wizard in this dialog by pressing the **Cancel** button.

Next and **Back** buttons can be used to move between Wizard dialogs.

Parameter Description

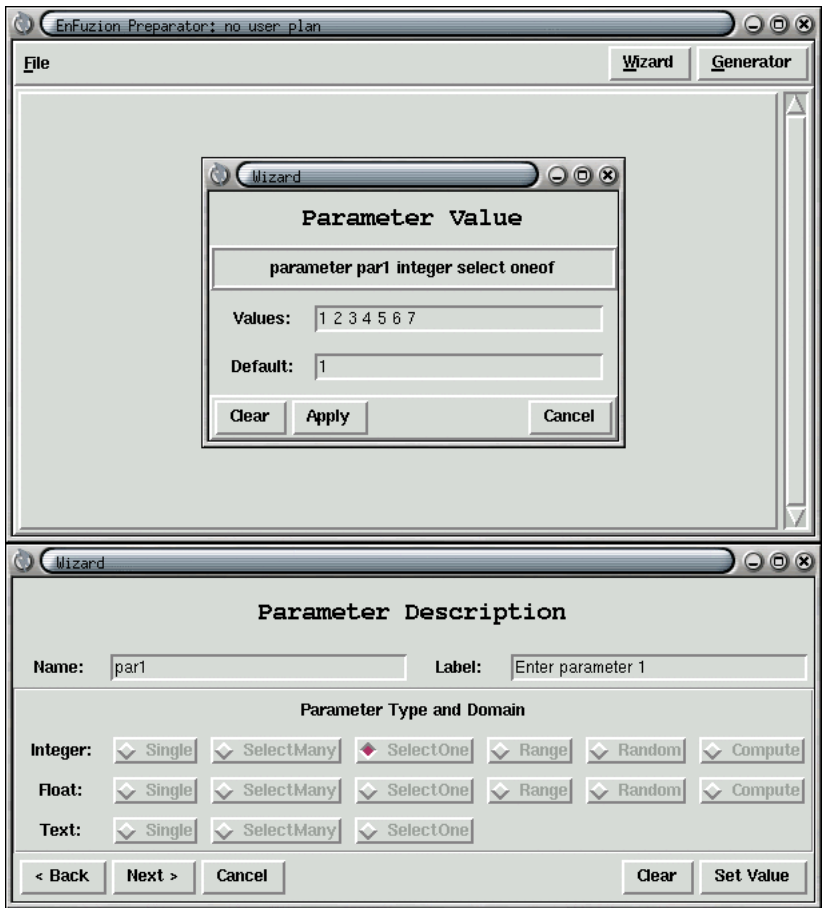
The Parameter Description dialog allows you to create parameter statements using a graphical interface. You can specify parameter type, domain, domain values and default values.

The **Set Value** button allows you to specify parameter values. It opens a Parameter Value dialog with fields for parameter values. The **Apply** button in the Parameter Value dialog will generate a plan statement for the specified parameter.

The **Clear** button clears all parameter fields.

A sample screen of the Parameter Description is shown in Figure 8-2. The figure shows a definition for the parameter "par1".

Figure 8-2. Preparator Description Dialog



The following parameter statement is generated as a result of this dialog:

```
parameter par1 label "Enter Parameter 1" \
       integer select oneof 1 2 3 4 5 6 7 default 1;
```

More details about different parameter types can be found in the Section called *Parameters*.

Preprocessing Dialog

A preprocessing command is a command which is to be performed on the root computer prior to starting any other jobs. A common use for preprocessing commands is to set up files for a run, delete old files, or perform other general preprocessing. Each preprocessing command is added to the plan by filling in the dialog entry and by pressing the **Apply** button.

Input Files Dialog

The Input Files dialog enables you to enter files which will be copied to each remote node before any of the user jobs are started on that node. Each input file is added to the plan by filling in the dialog entry and by pressing the **Apply** button.

Substitution Files Dialog

The Substitution Files dialog enables you to enter files that require substitution on a node. During substitution, the source file is copied to the destination file, and all parameter place holders are replaced with actual parameter values. It is assumed that both the source file and the destination file are on a node computer.

Each substitution is added to the plan by filling in the dialog entries and by pressing the **Apply** button.

User Commands Dialog

The User Commands enables you to specify user commands for execution on a node. Each command is added to the plan by filling in the dialog entry and by pressing the **Apply** button.

Output Files Dialog

The Output Files dialog enables you to specify files, which are copied back to the home directory on the root computer, after all the jobs have finished. Each file is added to the plan by filling in the dialog entries and by pressing the **Apply** button.

Post Processing Dialog

The Post Processing Command is a command which is to be performed on the root computer after all the jobs have finished. A common use for this is for cleaning up files, or for performing visualization of results. Each post processing command is added to the plan by filling in the dialog entry and by pressing the **Apply** button.

Finishing Dialog

This is the final wizard dialog. You can either confirm the data entered thus far by pressing the **OK** button, or go back to add more data by pressing the **Back** button.

A Sample Plan

The example below demonstrates how to use the Wizard to construct a plan. The plan is a generic example of the most common EnFuzion applications.

Preparing Input Files

First, the input files are generated by running command **makefiles**. Input files consist of the files **input1**, **input2** and **skeleton**. Files **input1** and **input2** are data files, and they do not require any changes. The file **skeleton** contains parameter place holders, which need to be replaced with actual values for each job.

Initializing the Nodes by Copying Input Files

After the files are generated, they are copied to the nodes.

Executing the Jobs

For each job, the parameter place holders in the file **skeleton** are replaced with actual parameter values. The result is stored in file **parameterfile**. Then, the user command **simulation** is executed by the node computer, taking **parameterfile**, **input1** and **input2** as input files and producing files **output1** and **output2** as output files. After the user command finishes, the resulting output files **output1** and **output2** are copied back to the root computer. The output files are renamed, so that their names do not conflict with output files from other jobs.

Post Processing of Output Files

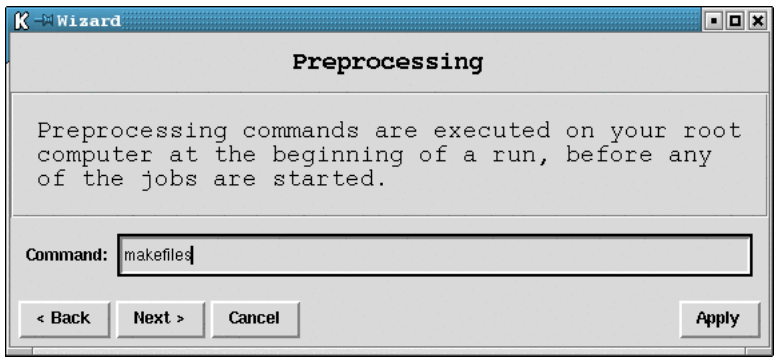
After all of the jobs finish, the output files are post processed on the root computer.

Below is a step by step guide that demonstrates how to use the wizard to compose an EnFuzion plan for the application detailed in the example above.

Step by Step Guide through the Wizard

During the preprocessing step at the beginning of an application, program **makefiles** will be run on the root computer to generate input files. Enter **makefiles** in the Preprocessing dialog and press **Apply**.

Figure 8-3. Entering a Preprocessing Command

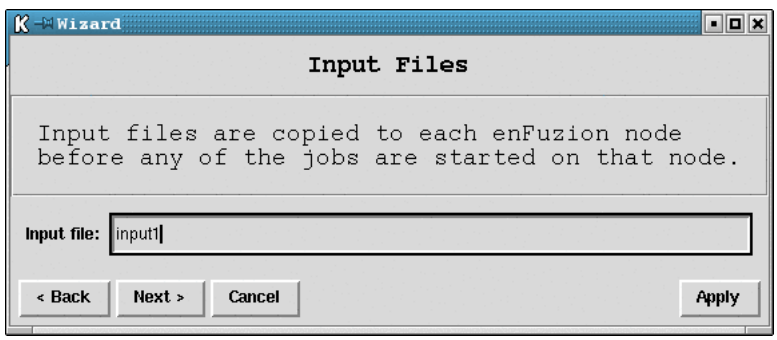


The following plan statements are generated by this dialog:

```
task rootstart
    execute makefiles
endtask
```

Each job requires three input files: **input1**, **input2** and **skeleton**. Enter each file separately into the Input Files dialog. Press the **Apply** button to generate the corresponding plan statement.

Figure 8-4. Entering an Input File

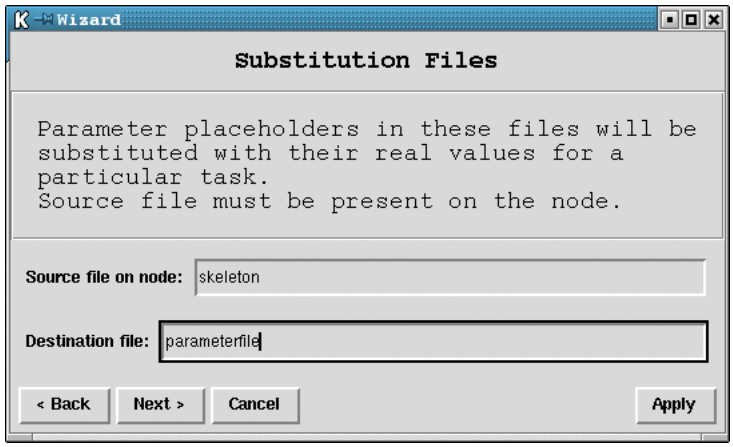


The following plan statements are generated to copy input files to all of the nodes:

```
task nodestart
    copy input1 node:.
    copy input2 node:.
    copy skeleton node:.
endtask
```

A parameter substitution is performed on a node computer for each job. The substitution takes the **skeleton** file as a source and produces the parameters file as a destination. To specify a substitution for a job, enter a source and a destination file in the Substitution Files dialog and press **Apply**.

Figure 8-5. Entering a Parameter Substitution

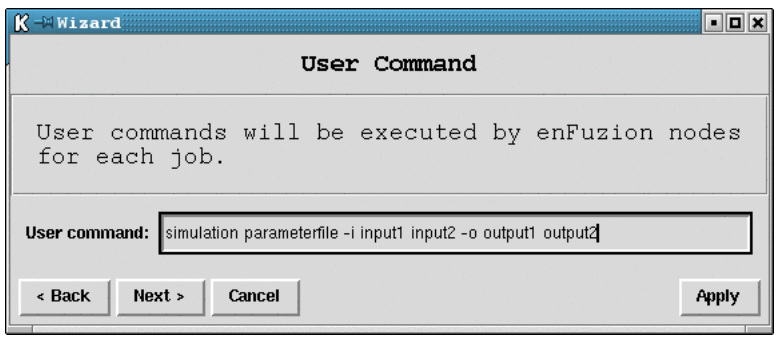


The following plan statements are generated by this dialog:

```
task main
    node:substitute skeleton parameterfile
endtask
```

The user command is a program called **simulation**. It is assumed that the executable for the program is already installed on the node. To specify a user command, enter its command line in the User Command dialog and press **Apply**.

Figure 8-6. Entering a User Command

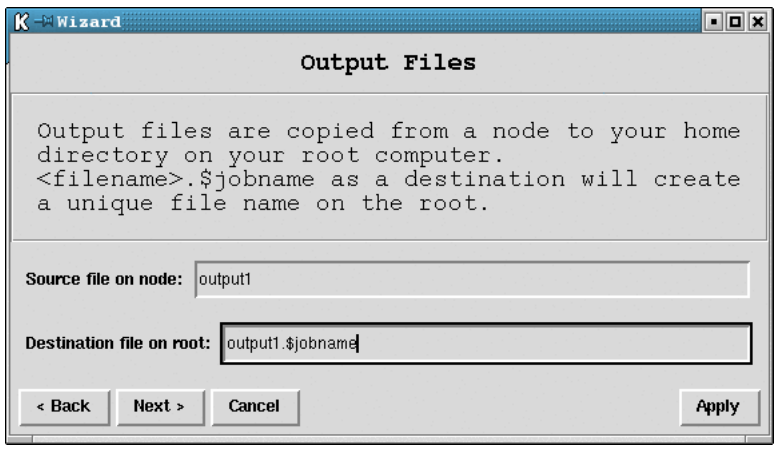


The following plan statements are generated by this dialog:

```
node:execute simulation parameterfile -i input1 input2 \
                -o output1 output2
```

Output files **output1** and **output2** are copied to the root computer after the main task finishes. On the root computer, the output file names are changed to include a job identifier using the $jobname parameter, as discussed in the Section called *Parameters*. This generates different output file names for each job. To specify an output file, for each of **output1** and **output2**, enter its source and destination name in the Output Files dialog and press **Apply**.

Figure 8-7. Entering an Output File

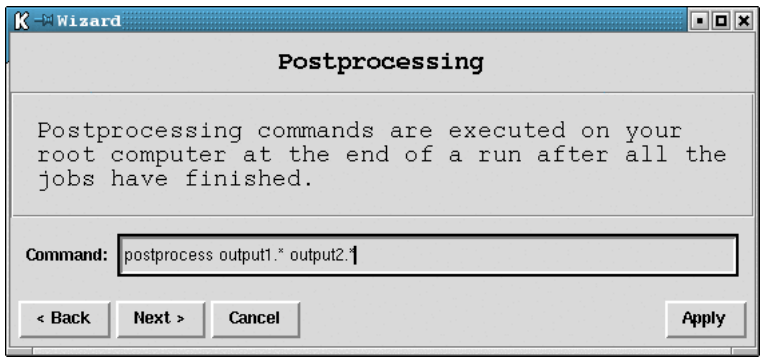

Wizard
Output Files
Output files are copied from a node to your home directory on your root computer.
<filename>.$jobname as a destination will create a unique file name on the root.
Source file on node:
output1
Destination file on root:
output1.$jobname
< Back
Next >
Cancel
Apply

The following plan statements are generated by this dialog:

```
copy node:output1 output1.$jobname
copy node:output2 output2.$jobname
```

Output files are processed after all of the jobs have finished. Post processing commands are executed after all jobs are finished. Enter "postprocess output1.* output2.*" in the Post processing dialog and press **Apply**.

Figure 8-8. Entering a Post Processing command


Wizard
Postprocessing
Postprocessing commands are executed on your root computer at the end of a run after all the jobs have finished.
Command:
postprocess output1.* output2.*
< Back
Next >
Cancel
Apply

The following plan statements are generated by this dialog:

```
task rootfinish
    execute postprocess output1.* output2.*
endtask
```

Figure 8-9 shows a complete plan, produced by the wizard. This plan can be saved to a file. The plan can be also edited in the main window before saving.

Figure 8-9. Sample Output Plan from Preparator

```
parameter par1 label "Enter Parameter 1" integer \
               select oneof 1 2 3 4 5 6 7 default 1;
task rootstart
    execute makefiles
endtask
task nodestart
    copy input1 node:.
    copy input2 node:.
    copy skeleton node:.
endtask
task main
    node:substitute skeleton parameterfile
    node:execute simulation parameterfile -i input1 input2 \
                    -o output1 output2
    copy node:output1 output1.$jobname
    copy node:output2 output2.$jobname
endtask
task rootfinish
    execute postprocess output1.* output2.*
endtask
```

Specifying Input Values

Input values can be entered through a graphical program, called the Generator. The Generator takes a plan file, containing job templates and a description of parameters. It produces an application specific graphical user interface, which is used to select parameter values. After the parameters values are selected, the Generator produces a run file, which contains a complete description of jobs and parameter values for each job. The run file is submitted to the Dispatcher for execution.

The sections below describe the Generator and provide an example of an application specific graphical user interface.

The Generator

The Generator can be started on a command line as:

```
enfgenerator [ -g ] [ <plan_name> ]
```

Normally, the Generator is run in interactive mode with a graphical user interface, described next. However, if the option **-g** is specified on the command line, the Generator will be executed with no graphical interface in batch mode. This mode is useful for calling the Generator directly from other programs, or even from other EnFuzion jobs.

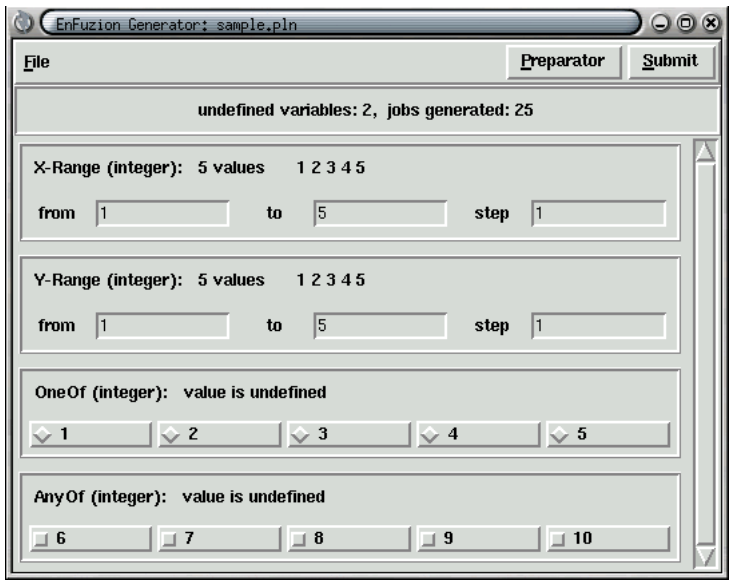
In batch mode the Generator will take a plan file as input and automatically produce a run file. In batch mode, the Generator expects that all variables have their default values set. Otherwise, an error is reported. The default values can be set with a prior interactive execution of the Generator or with the Preparator.

If the **-g** option is not specified, the Generator displays an application specific, graphical interface for specifying the values of input parameters. The interface allows you to change existing parameter values. It updates the current values and shows the number of generated jobs.

A Sample Application Specific Graphical User Interface

This section provides a sample, application specific interface, including how it is used to specify input values and generate jobs. An example of this interface is shown in Figure 8-10.

Figure 8-10. Application Specific Interface in Generator

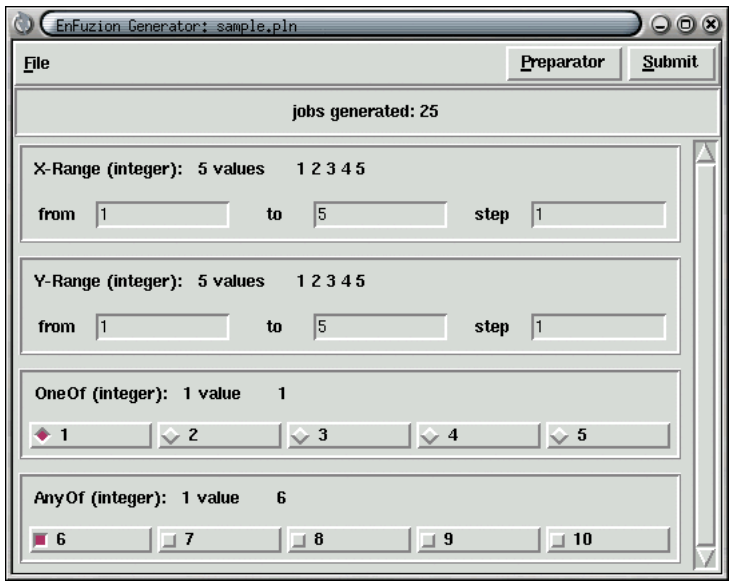

EnFuzion Generator: sample.pln
File
Preparator
Submit
undefined variables: 2, jobs generated: 25
X-Range (integer): 5 values 1 2 3 4 5
from 1 to 5 step 1
Y-Range (integer): 5 values 1 2 3 4 5
from 1 to 5 step 1
OneOf (integer): value is undefined
1 2 3 4 5
AnyOf (integer): value is undefined
6 7 8 9 10

The interface demonstrates graphical constructs for the different type domains: single value, range, select anyof, select oneof, and random. It shows that the values of two of the parameters are still undefined. The values, that are already filled in, have been already specified as defaults in the plan file. All default values can be changed in the Generator to obtain the final values. The interface has been created from a plan file with the following parameters, where parameters oneof and anyof do not have a default input value:

```
parameter xrange label "X-Range" integer range;
parameter yrange label "Y-Range" integer range;
parameter oneof label "OneOf" integer select oneof 1 2 3 4 5;
parameter anyof label "AnyOf" float select anyof 6 7 8 9 10;
```

After the values for the two undefined parameters are specified, the interface updates the values and the number of jobs is generated. An example of an updated interface is shown in Figure 8-11.

Figure 8-11. Interface with All Parameters Defined


EnFuzion Generator: sample.pln
File
Preparator
Submit
jobs generated: 25
X-Range (integer): 5 values 1 2 3 4 5
from 1 to 5 step 1
Y-Range (integer): 5 values 1 2 3 4 5
from 1 to 5 step 1
OneOf (integer): 1 value 1
1 2 3 4 5
AnyOf (integer): 1 value 6
6 7 8 9 10

At any time, the current parameter values can be saved with new defaults in a plan file. These commands are under the menu **File**, using one of the options **Save Plan** or **Save Plan As...** .

After no undefined parameters are left, the existing description of jobs can be saved to a run file. These commands are under the menu **File**, using one of the options **Save Run** or **Save Run As...** .

After the run file is created, it can be submitted to the Dispatcher for execution, using the **File** menu, option **Submit**, or the button **Submit** in the toolbar. The Generator exits after a run submission. The output from the submission process is stored to files **stdout.txt** and **stderr.txt**. These files are useful to find the run ID and any error messages. Run submission and execution is described in more detail in Chapter 9.

If you want to change the plan by adding new parameters or task statements, you can return to the Preparator with the option **Preparator** under the **File** menu or the button **Preparator** in the toolbar. The plan file can be also edited with a standard text editor.

Description of Plan Files

Plan files specify user jobs to be distributed over the network and executed. Plans contain a description of the various parameters and instructions on how to execute user programs and perform file transfer. Plans consist of several main sections. These main sections are parameters, tasks, and configuration options.

Parameters in plan files are provided as templates, which define possible input values. These templates help users in selecting specific input values for a run. Parameter templates in plan files are defined with the parameter statement.

Tasks specify the commands that are executed during the execution of each job in a run. The number of tasks in a run is not limited. All jobs in a run share the same tasks. Some task names and their functions are predefined. Tasks are defined with the task statement.

Configuration options can be used to change default values for EnFuzion provided variables. Configuration options are defined with the set statement.

The following sections give details on parameters, tasks and configuration options.

Comments

Lines in a plan file can contain comments. Comment lines begin with a "#" character and end with an end of line. These lines are ignored by EnFuzion.

Parameters

Parameters are used to define input values for jobs. In general, each job receives a different set of input parameter values. The selection of input parameter values for each job and generation of jobs is done by the Generator, which takes a plan file and produces a run file.

Parameters are defined in parameters statements and used in task commands. Task commands, described in the Section called *Task Commands*, contain parameter macro placeholders, which are replaced with job specific parameter values the job execution time.

The substitution of parameter placeholders is specified in more detail in the Section called *Parameter Substitution*. The sections below provide details on the definition of parameters and the parameter statement.

The Parameter Statement

The Parameter Statement defines a parameter. The most important aspects of a parameter are its name, type and domain. The name identifies the parameter macro placeholder in task commands. The type can be an integer, a floating point number or a text. The domain specifies the method, by which multiple values for the parameter are generated, such as a range, random, a user specified range of values and so on.

The parameter statement has the following syntax:

```
parameter <name> [<label>] <type> [<domain>];
```

The required field <name> identifies the parameter. Parameter names must start with an alphabetic character: "a-z,A-Z", followed by any alphanumeric character: "a-z,A-Z,0-9" or the "_" character. For example, the following names are valid:

```
variable, PAR1, PAR_11_0
```

If the optional <label> is specified, it will be used by the Generator to label the parameter. If the label is not specified, the parameter <name> will be used by default. The label value must be enclosed in double quotes. An example of a legal <label> field is:

```
"Enter Value"
```

The required field <type> specifies the parameter type. Supported parameter types are integer, float or text. Any value for a text parameter must be enclosed in double quotes. The type is used to verify user input by the Generator while input value are being specified. All parameters are handled as text during execution by the Dispatcher.

The optional field <domain> determines the method, by which multiple values for the parameter are generated. If the domain is not specified, the parameter will have a single value. The following domains are supported: single value, range, select anyof, select oneof, random, and compute. The domains are described in detail below.

- single value: the parameter has a single value. This domain is assumed, if no other domain is specified. The syntax for the single value domain is:

  ```
  [ default <value> ]
  ```

  The default value can be specified in the Preparator. The final value is specified in the Generator.

- range: range generates values between a lower and an upper bound. The syntax for the range domain is:

  ```
  range [from <value>] [to <value>] [points <value>]
  range [from <value>] [to <value>] [step <value>]
  ```

  Range can either generate a fixed number of uniformly distributed points or it can use a step value. Default values can be specified in the Preparator. Final values are specified in the Generator.

  Examples:

  ```
  parameter var5 integer range from 0 to 100 points 3;
  parameter var6 integer range from 0 to 10 step 2;
  parameter var4 integer range;
  ```

- select anyof: any combination of values in the value list can be selected. The syntax for the select anyof domain is:

  ```
  select anyof <value_list> [default <value_list>]
  ```

  Default values can be specified in the Preparator. Final values are specified in the Generator.

  Examples:

  ```
  parameter var1 integer select \
                 anyof 1 2 3 4 5 6 7 8 9 0 default 1 5 8;
  parameter var2 text select \
                 anyof "Mon" "Tue" "Wed" "Thu" "Fri" \
                 "Sat" "Sun" default "Sat" "Sun";
  ```

- select oneof: one of the values in the value list can be selected. The syntax for the select oneof domain is:

  ```
  select oneof <value_list> [default <value>]
  ```

Default values can be specified in the Preparator. The final value is specified in the Generator.

Examples:

```
parameter var3 integer select \
               oneof 1 2 3 4 5 6 7 8 9 0 default 1;
```

- random: random numbers are generated between a lower and an upper bound. The syntax for the random domain is:

  ```
  random [from <value>] [to <value>] [points <value>]
  ```

  Default values can be specified in the Preparator. Final values are specified in the Generator. Random supports two ways of generating random values, depending on the points option.

  If the value of the points option is a nonzero, positive integer, then random numbers will be generated before any jobs are produced. In this case, the points value influences the number of jobs generated. Each random value can be used by multiple jobs.

  Examples:

  ```
  parameter var6 float random from -1.0 to 1.0 points 6;
  ```

  This example creates 6 random numbers between -1.0 and +1.0. These are assigned to jobs as if created by a range option.

  If the points option is omitted or its parameter is 0, new random numbers will be created independently for each job during the generation of jobs. In this case, this parameter will not increase the number of jobs, and each random value will be used by one job only.

  Examples:

  ```
  parameter var7 float random from -1.0 to 1.0;
  ```

  This example creates a new random number between -1.0 and +1.0 for every job that is generated.

- compute: the value is computed from values of other parameters. The syntax for the compute domain is:

  ```
  compute <expression>
  ```

  The expression can contain only operators that specified by the Preparator.

  Expressions can contain standard arithmetic operators "+", "-", "*" and "/". The "%" (mod) operator is also available for integer parameters. Supported standard functions are: ln - natural logarithm, log10 - log base 10, exp - power of e, exp10 - power of 10, sqrt - square root. These can be used to generate logarithmic, exponential and square root distributions.

  Examples:

  ```
  parameter var8 float compute sqrt(var1)*sqrt(var1);
  ```

EnFuzion Defined Parameters

EnFuzion defines a large set of parameters, which can be used in plan files. These parameters are discussed in detail in the Section called *Variables*.

Tasks

A task specifies the commands that are executed during the execution of each job in a run. The number of tasks in a run is not limited. All jobs in a run share the same tasks. Some task names and their functions are predefined.

The Task Statement

The Task Statement defines a task. Each task description starts with the task statement, followed by command statements, and ends with the endtask statement. Statements can span more than one line by placing the continuation character "\" at the end of each line to be extended.

The task statement has the following syntax:

```
task <name>
    <statement>
    <statement>
    ........
    ........
    <statement>
endtask
```

The <statement> can be either a task command or a conditional statement, described below.

Predefined tasks

Some task names are predefined by the system and have a special meaning. These are: **rootstart**, **nodestart**, **main**, **rootfinish**, and **onerror**. Predefined tasks allow you to specify commands to be executed at each phase of execution. These tasks support the four phases of EnFuzion execution: *root startup*, *node startup* , *job execution*, and *root completion*. For example, in **rootstart** you might wish to create some files when the EnFuzion root is started, copy those files to each node as it starts in **nodestart**, and then run a specific application for each job in **main**.

rootstart

The **rootstart** task is executed at the beginning of a run. No other jobs are started before this task completes successfully.

rootfinish

The **rootfinish** task is executed at the end of a run after all the user jobs are done, either successfully or with a failure.

nodestart

The **nodestart** task is executed on each node after the rootstart task completes, but before any user jobs are executed on the node. This task is used for customized node initialization.

main

The **main** task specifies the execution of the main user job. This is the default task for user jobs.

onerror

The **onerror** task can be specified to control which commands are executed when an error arises during job execution. The **onerror** task is executed whenever a command executed on a node returns non-zero exit status.

By default, if the **onerror** task is not specified, all node files are copied to the root host into directory **error.$ENFJOBNAME**.

Take the following task file:

Example:

```
task main
    node:execute mycommand
endtask
task onerror
    copy node:*.log error.$ENFJOBNAME
endtask
```

Here, the 'onerror' task is executed in case 'mycommand' returns an error. The task copies all files ending with **.log** from node to root, into directory **error.$ENFJOBNAME**.

Parameter Substitution

Parameters provide input values for jobs in the run. Parameters can be EnFuzion or user defined. System parameters are predefined and described in the Section called *Parameters*. In task statements, the use of EnFuzion provided parameters is the same as the use of user provided parameters.

Parameter values are substituted at run time by means of macro placeholders. These placeholders can be used in task commands and specify the location for substitution of parameter value. When a placeholder is found in a task description or in a substitution file, it is replaced at run time with the parameter value for that job.

Parameter placeholders are denoted by the "$" character followed by the parameter name. If a "$" character is not followed by a valid parameter name, then no substitution is performed. This allows free use of the "$" character.

Substitution can be avoided by using two "$" characters. For example, even if there is a parameter called variable, in the line below:

```
execute echo $$variable
```

the substitution process removes only the first "$" and produces the following line, which is executed by the job:

```
execute echo $variable
```

A parameter can be embedded in another string by surrounding it with pointed braces, "{" and "}". For example, if the value of parameter 'num' is 1, then the following parameter placeholder:

```
HELLO${num}THERE
```

yields the value:

```
HELLO1THERE
```

Locators

Locators are a common element of task statements. They specify local or remote hosts. They can be used in task commands and file descriptions. For task commands, they specify the host to execute the command. For files, they specify the location of the file. Locators can be any of the following:

root, node, local, remote

Locators 'root' and 'node' are absolute addresses, specifying the root and the node host for each job; 'local' and 'remote' are relative addresses. On the root host, 'local' is the root host and 'remote' is the node host. On the node host, the meaning is reversed.

If a locator is omitted, its default value is 'local' for files and 'root' for commands.

Task Commands

Task statements consist of EnFuzion commands. The following commands are supported:

cd, **checkfile**, **checksize**, **copy**, **execute**, **limit**, **loadparameters**, **mkdir**, **onerror**, **options**, **server**, **set**, **sleep**, **substitute**, **unset**

*Command **cd***

The syntax of the **cd** command is:

```
cd <directory>
node:cd <directory>
```

The **cd** command changes the current working directory, either on the root or on the node. The new working directory must exist and must be accessible to EnFuzion, otherwise the command returns an error.

*Command **checkfile***

The syntax of the **checkfile** command is:

```
checkfile <file>
node:checkfile <file>
```

The **checkfile** command verifies that a file or a directory path exists, either on the root or on the node. If the file or the directory exists, the command has no effect and returns successfully. If the file or the directory does not exist, the command fails. For **<file>** descriptions with wild-card characters, the command is successful, if at least one file matches the wild-card description. If no files match the wild-card description, then the command fails.

The command is also useful in combination with a **copy** command that contains wild-card characters. When placed just before the **copy** command, the **checkfile** command verifies that there is at least one file available for the **copy** command.

*Command **checksize***

The syntax of the **checksize** command is:

```
checksize <size> <file>
node:checksize <size> <file>
```

The **checksize** command verifies that a file is larger than <size> bytes. If the file contains more than <size> bytes, the command has no effect and returns successfully. If the file is smaller, the command fails.

The command can be useful for checking that programs produce expected results. In case of a problem, programs often do not signal an error, but complete successfully and produce a short error message instead of a longer output file.

*Command **copy***

The syntax of the command is:

```
copy <source_file> <destination_file>
```

The **copy** command copies a source file to a destination file. Both the source file name and the destination file name can include a locator, which describes the host location for the file. For example: **root:input** is file input on the root host, and **node:output** is file output on a node host. If no locator is specified, then the default value is **root**.

On the root, files are located in the **run** subdirectory. On the node, files are located in the **job** subdirectory. If the source file is on the root and is not found in the **run** subdirectory, then the parent directory is searched for the file.

<source_file> can contain wild card constructs: "*", "?" and "[...]". "*" matches any number of characters, "?" matches one character, and "[...]" matches any of the characters inside the square brackets. If any of the files is a directory, the entire directory tree is recursively copied.

If the <destination_file> exists, it is cleared first. If the <destination_file> is a directory name, source files are copied into the directory. If the <destination_file> is a ".", this denotes the same name as the <source_file>. The **copy** command creates any necessary directories for the <destination_file>.

Examples:

```
copy root:input node:.
```

The example above copies the file **input** from the root to the node host, and names the file **input**.

```
copy root:input/* node:.
```

The example above copies every file from the root host directory called **input** to the node host.

```
copy root:input/* node:input
```

The example above copies all of the files in the **input** directory to a directory called **input** on the node.

A single **copy** command can copy more than one file. In that case, all files are copied to the same destination directory.

Example:

```
copy input1 input2 input3 node:.
```

The example copies files from the root to the current directory on the node host. File names are not changed.

The **copy** command supports option **-t**, which specifies that all files are copied as text files. Files are converted from Unix to Windows format or vice versa depending on the operating system of the source and destination hosts.

For example:

```
copy -t root:input.txt node:.
```

This command converts the file **input.txt** from Unix to Windows text format if the root host is Linux and the node host is Windows.

When using the **-t** option, all files specified by the <source_file> argument must be text files. Binary files must be copied without the **-t** option. Special care must be taken when using wild card expansions.

*Command **execute***

The syntax of the command is:

```
execute <user_command>
```

The execute command runs <user command>. EnFuzion expects that <user_command> returns 0 as its terminating value, if it completes successfully. Any other return value is treated as an error, and by

default, it terminates the job with a fail status. Default handling of non-zero return values can be changed with the onerror command.

On Unix, <user_command> is passed to shell **/bin/sh**, so it can contain shell constructs.

On Windows, the execution of <user_command> depends on shell redirection characters: ">", "<" and "|". If <user_command> does not contain any shell redirection characters, the command is executed directly. If <user_command> contains shell redirection characters, then the command is passed to the command processor cmd or as specified by the **%ComSpec%** environment variable. In this case, programs executing on EnFuzion Windows nodes cannot be terminated by EnFuzion, although this might be specified in EnFuzion options.

The locator specifies the host to execute the <user_command>. For example, root:execute executes the <user_command> on the root host. And node:execute executes the <user_command> on the node host. If no locator is specified, the command is executed on the root host by default.

The <user_command> can use standard input, standard output and standard error. These can be either redirected to files with shell redirection constructs or left unspecified. If they are unspecified, EnFuzion automatically redirects standard input to file **stdin**, standard output to file **stdout**, and standard error to **stderr**. These regular files can be handled with standard commands for file manipulation.

Examples:

```
node:execute ls
```

The example above executes the **ls** command on the node host.

```
node:execute simulation
```

The example above executes the program **simulation** on the node host.

*Command **limit***

EnFuzion implements a wide range of timing options. These options provide flexible error handling, which can be adapted to different application needs. The options allow users to fine tune EnFuzion operation for their particular environment.

The syntax of the **limit** command is:

```
limit connect <time>
limit request <time>
limit compute <time>
limit complete <time>
limit idle <time>
```

Time is specified in one of the following three forms:

```
hh          hours
hh:mm       hours, minutes
hh:mm:ss    hours, minutes, seconds
```

Options to the **limit** command have the following meaning:

connect

This option applies only to datajobs. Time that the node command waits for the user program to connect for the first time. This is the maximum time that the user program is allowed for initialization. The default value is unlimited.

request

This option applies only to datajobs. Time that the node command waits for the next datajob request after a result is received. This is the maximum time that the user program is allowed for the processing between datajobs. The default value is unlimited.

compute

This option applies only to datajobs. Time that the node command waits for a result after the input is passed to the user program. This is the maximum time that the user program is allowed for the processing of one datajob. The default value is unlimited.

complete

This option applies to jobs and datajobs. Total time that the execute or server commands are allowed to use. This is the maximum time that the command is allowed to execute. The default value is unlimited.

idle

This option applies only to jobs. Time that the user program is allowed to be idle and not consuming any CPU cycles. If the user program is idle longer, it is terminated by EnFuzion with an error. Any dialog Windows by the user process will be captured and shown in the run log. This option is implemented only on Windows based systems.

Examples:

```
limit complete 00:00:30
```

The example above terminates execution of each user program that does not complete in 30 seconds.

```
limit connect 00:05:00
```

The example above terminates execution of a user program, started with the node command, that does not connect to EnFuzion in five minutes.

*Command **loadparameters***

The syntax of the **loadparameters** command is:

```
node:loadparameters <file>
```

The **loadparameters** updates parameters with values from <file>. <file> must contain parameter values in the form:

```
<name>=<value>
```

<name> specifies parameter name and <value> specifies parameter value. Each line in <file> contains one parameter.

The command reads <file>, updates parameter values, and substitutes all task commands with new values. The execution continues with the next command in the task. Previously executed task commands are not repeated.

Command ***mkdir***

The syntax of the **mkdir** command is:

```
mkdir <directory>
node:mkdir <directory>
```

The **mkdir** command creates a directory, either on the root or on the node. If a directory already exists, the command has no effect and returns success.

Command ***onerror***

The command **onerror** specifies the handling of execution user errors. These can be caused by errors, such as missing files or user programs, that return non-zero exit status. The syntax of the **onerror** command is one of the following:

```
onerror fail
onerror repeat
onerror restart
onerror ignore
```

onerror fail causes the job to fail, when an error is encountered. This is the default option.

onerror repeat submits the job back to the execution queue, when an error is encountered. The job will not be sent for execution on the same node more than once. This option is useful when jobs have different sizes and some hosts are incapable of running all of the jobs.

onerror restart submits the job back to the execution queue, when an error is encountered. The job can be sent for execution to the same node more than once, depending on node availability. This option can be used when jobs fail due to lack of resources, which become later available.

With the **onerror ignore** option, all errors are ignored. All task commands are executed and the task completion is counted as successful, regardless of any user errors.

Error handling can also be controlled via **task onerror** as described in the Section called ***onerror***. For additional information on error handling see the Section called *Handling of Network Failures* in Chapter 1.

Command ***options***

The **options** command loads a new set of load monitoring options on the node where it is executed.

The syntax of the **options** command is:

```
node:options <options_file>
```

<options_file> file can be on the root or on the node host. It must contain EnFuzion node options as would be specified in an **enfuzion.options** file. See Chapter 7.

Examples:

```
node:options options_file
```

This example copies the **options_file** file from the current directory on the root to the node and loads a new set of node monitoring options.

```
node:options node:new_options
```

This example takes the **new_options** file in the current directory on the node and loads a new set of node monitoring options.

*Command **server***

The **server** command allows user programs on nodes to communicate directly with EnFuzion. This can significantly speed up job processing by eliminating the requirement to handle input data or job results through files.

The **server** command manages the flow of inputs and results. It communicates with the Dispatcher on the root and with the user program on the node host. Jobs for the server command are lightweight jobs, called datajobs . Input for datajobs consists of a string of input data. The size of the string is limited to 20Kb by default, in order to prevent excessive input sizes. The default size can be increased by changing the run variable ENFMAXDATASTREAM. The input data is sent to the user program, which returns a string of data as a result. The result is returned back to the Dispatcher, which communicates the result to the user. Datajobs have no associated task in the run file, since all the processing is done by the user program. User programs utilize EnFuzion capabilities for load balancing, error handling, and data routing. The server command completes when there are no more datajobs to process.

The syntax of the **server** command is:

```
node:server [ -b <size> ] [ -p <port> ] [ <user command> ]
```

Option **-b** specifies the size of the buffer. EnFuzion bundles <size> datajobs in one message, which can reduce processing time. The maximum limit for the buffer size is 1000 datajobs. The optimum size of the buffer is dependent on each particular environment.

Option **-p** specifies the port of the user program on the node host to which EnFuzion connects. For each datajob received, EnFuzion connects to the user program, sends the datajob input to the program, receives the output and closes the connection. EnFuzion appends newline and null characters ("\n\0") after the input. The user program can use them as terminators of the message. EnFuzion terminates output from the user program, when it reads the end of file.

If **-p** option is not specified, EnFuzion uses <user command> to start the user program, waits for a connection from the program, and then proceeds as with the **-p** option. When there are no more datajobs to process, the user program is terminated and the server command completes. The user program must connect to EnFuzion via a local socket on Unix based systems or via a local pipe on Windows based systems. The name of the local socket or pipe is passed to the user program in the ENFSOCKET environment variable.

Examples:

```
node:server addnumbers
```

The example above starts a user program called **addnumbers**, and uses it to process datajobs. When all the datajobs are processed, **addnumbers** is terminated and the command completes.

```
node:server -p 1234
```

The example above requests that EnFuzion on the node connects to the user program via port "1234" and uses the user program to process datajobs. When all the datajobs are processed, the command completes.

```
node:server -b 500 -p 1234
```

This is the same as the previous example, except that 500 datajob inputs are received at once by the node. In general, this can significantly speed up job processing.

Run has a limit variable ENFDATASTREAM_EXECUTION_LIMIT. If a datajob does not provide a result within the limit, the datajob is rescheduled. The default value is unlimited.

*Command **set***

The **set** command sets a variable value. This allows jobs to change variable values dynamically at runtime. This capability complements the ability to set variables through the EnFuzion API or statically in configuration or run files. The variable is defined locally in the scope in which the command executes. It is not visible outside of the scope.

The syntax of the **set** command is:

```
set [ <scope> ] <name> [ <value> ]
```

<scope> can be any of the following:

- **-cluster**

  The variable is global, available to all jobs.

- **-run**

  The variable is local to the run. It is available only to the jobs within the current run.

- **-node**

  The variable is local to the node. It is available to all jobs, executing on this node. This scope is not supported in ’rootstart’ and ’rootfinish’ tasks.

- **-context**

  The variable is valid for the node within the current run. It is available only to the jobs within the current run executing on that node. This scope is not supported in ’rootstart’ and ’rootfinish’ tasks.

- **-job**

  The variable is local to the job. It is available to all commands, executing within this job.

If <scope> is omitted, the default is "-job".

<name> specifies the variable name.

<value> is handled differently for single values and for lists, such as property lists. If the variable is a single value, then <value> specifies the new variable value. If <value> is omitted, a single value variable is set to an empty string. If the variable is a list, the <value> is added as an element to the list. If <value> is omitted, the command has no effect on the value of the list variables.

Example:

```
set nodetype MonteCarlo
```

The example above defines a job variable ’nodetype’ with value "MonteCarlo".

```
set -server ENFPROPERTY blue
```

The example above adds the value "blue" to ENFPROPERTY.

*Command **sleep***

The syntax of the **sleep** command is:

```
sleep <integer>
node:sleep <integer>
```

The **sleep** command waits <integer> seconds before the execution proceeds with the next command. It is useful for testing and debugging purposes.

*Command **substitute***

The syntax of the **substitute** command is:

```
substitute <source file> <destination file>
```

The **substitute** command substitutes all parameter placeholders in a <source file> with actual parameter values and produces a <destination file>. Parameter placeholders are specified in the the Section called *Parameter Substitution*. If the <destination file> exists, it is cleared first.

The locator specifies the host for the substitution. For example, root:substitute performs the substitution on the root host; node:substitute performs the substitution on the node host. If no locator is specified, the substitution is performed on the root host by default.

The <source file> and the <destination file> must be on the same host, both on the root host or both on the node host.

Examples:

```
substitute skel par
```

The example above copies file **skel** on the root host to file **par** and replaces any parameter placeholders with actual parameter values.

```
node:substitute skel par
```

The example above copies file **skel** on the node host to file **par** and replaces any parameter placeholders with actual parameter values.

Command ***updatefile***

The syntax of the **updatefile** command is:

```
updatefile <source file> <destination file>
```

The **updatefile** command is used to incrementally copy new file content from the node to the EnFuzion root. The command can be used to access files, such as log files, while a user job is executing. <source file> specifies a source file on the node and <destination file> specified a destination file on the EnFuzion root.

While a user job is executing on the node with the **execute** command, EnFuzion is checking <source file> for new content approximately every 15 seconds. If new content is found, it is added to <destination file> on the EnFuzion root.

Examples:

```
updatefile log.txt $ENFJOBNAME/log.txt
```

The example above appends new content from the **log.txt** on the node host to **$ENFJOBNAME/log.txt** on the root host approximately every 15 seconds.

Command ***unset***

If the variable is a single value, then the **unset** command deletes the variable from the defined variables. If the variable is a list, then the **unset** command deletes <value> from the list. If the variable is a list and no <value> is defined, the variable is deleted. If the variable is defined internally by the system, the command has no effect.

The syntax of the **unset** command is:

```
unset [ <scope> ] <name> [ <value> ]
```

<scope> can be any of the following:

- **-cluster**

  The variable is global, available to all jobs.

- **-run**

  The variable is local to the run. It is available only to the jobs within the current run.

- **-node**

  The variable is local to the node. It is available to all jobs, executing on this node. This scope is not supported in ’rootstart’ and ’rootfinish’ tasks.

- **-context**

  The variable is valid for the node within the current run. It is available only to the jobs within the current run executing on that node. This scope is not supported in ’rootstart’ and ’rootfinish’ tasks.

- **-job**

  The variable is local to the job. It is available to all commands, executing within this job.

If <scope> is omitted, the default scope is "-job".

<name> specifies the variable name.

Example:

```
unset -run nodetype
```

The example above removes variable ’nodetype’ from the run.

```
unset -node ENFPROPERTY green
```

Assuming that ENFPROPERTY contains "red", "green" and "blue" before the unset command is issued, it contains only "red" and "blue" afterward.

Conditional Statements

EnFuzion run files can contain conditional statements in addition to the task commands described above. These statements allow runtime selection of several execution alternatives. The syntax of conditional statements is:

```
if ( <condition> ) then
    <statement>
    ...
    <statement>
else if ( <condition> ) then
    <statement>
    ...
    <statement>
else
    <statement>
    ...
    <statement>
endif
```

Conditions in the statement are evaluated from the start. When the first <condition> evaluates to true, its block of statements is evaluated. Remaining statements are skipped. If none of the conditions evaluates to true, then the else part without a condition is executed, if it is present. The else part is not required.

The <condition> can be one of the following:

- **<string1> == <string2>**

  Both strings are compared. The condition returns true, if they are the same. Otherwise, it returns false. A string can be also a job parameter.

- **<string1> != <string2>**

  Both strings are compared. The condition returns true, if they are different. Otherwise, it returns false. A string can be also a job parameter.

- **-e <file_name>**

  The condition returns true, if the file or the directory <file_name> exists on the node host. Otherwise, it returns false.

- **-m <file_name>**

  The condition returns true, if the file or the directory <file_name> is missing on the node host and does not exist. Otherwise, it returns false.

An example of a conditional statement is:

Example:

```
if ($ENFOS == "WindowsNT") then
    node:execute echo "This is a Windows NT machine"
else if ($ENFOS == "Linux") then
    node:execute echo "This is a Linux machine"
else if ($ENFOS == "Darwin") then
    node:execute echo "This is a Mac OS X machine"
else
    node:execute echo "This is a " $ENFOS " machine"
endif

if (-m /home/user/input) then
    # copy input file from the node
    copy input node:/home/usr/input
    node:execute echo "Input file was copied"
else
    node:execute echo "Input file already exists and was not copied"
endif
```

Commands from External Scripting Languages

External scripting languages can be easily integrated with task commands. Examples of external scripting languages are command shells, such as sh, or csh, and interpreted languages, such as Perl, Python, Ruby and Tcl.

External scripts can be called by using the **execute** command.

Example:

```
node:execute <script_command>
```

Script commands can use the full power of EnFuzion task commands by calling the Enfexecute program. The Enfexecute program is provided with EnFuzion.

Program Enfexecute

The program Enfexecute takes a task command as a command line parameter and executes that command. See the Section called *Task Commands*.

```
enfexecute 'task_command'
```

Enfexecute can be called from any program or scripting language. Its command line can contain any parameter values, defined for the job that is calling **enfexecute**. Parameter values are passed from EnFuzion through the environment. If any of the parameter values is required, the program that calls **enfexecute** must make sure that the environment is passed along.

The following is an example of the use of **enfexecute**. A simple task is:

```
task main
    node:execute compute >output
    copy node:output output.$ENFJOBNAME
endtask
```

A similar functionality is obtained by using **enfexecute**, embedded in an **sh** script. The task is:

```
task main
    node:execute script.sh
endtask
```

The contents of **script.sh**:

```
#!/bin/sh
compute >output
enfexecute copy node:output output.$ENFJOBNAME
```

Configuration Options

EnFuzion configuration options are predefined variables, which can be used to modify default EnFuzion behavior. Details on configuration options and variables in general are provided in the Section called *Variables*.

Values for configuration options can be set with the **set** statement, described in the next section.

The Set Statement

The **set** statement sets a variable value.

The syntax of the **set** statement is:

```
set <name> <value>;
set <name> "<value>";
```

The <value> can be without quotes when it is a simple integer, otherwise it must be enclosed in quotes.

Example:

```
set ENFPRIORITY_LEVEL 60;
set ENFNOTIFY_ADDRESS "bob@company.com";
```

This example raises the run priority from default 50 to 60 and sets the notification address to "bob@company.com".

Including Contents from Other Files

Contents from other files can be included in a plan file with the include statement:

```
include "<filename>"
```

Description of Run Files

The run file describes a run. It can be directly submitted for execution. The run file contains definitions of jobs, tasks and variables. These definitions can be specified in any order.

Tasks and variables are the same as in plan files. Check out the Section called *Tasks* and the Section called *Configuration Options* for details. Details on job descriptions are provided in the next few sections.

Jobs

Jobs specify the work to be done by nodes. If required, a job can access the root for root related operations, such as copying output files back to the root machine. Jobs are generated by the Generator or can be specified directly in a run file. Another option is to define jobs dynamically through an EnFuzion API command.

Job parameters provide job specific values during job execution. Each parameter value is a string. Values are used implicitly during parameter substitution in tasks or explicitly with the **substitute** command. See the Section called *Parameters*. Parameter values are also set in the execution environment on nodes. User applications can use environment variables to access parameter values.

Jobs can also have job specific variables. See the Section called *Variables*, for a description of job specific variables.

Jobs definitions are specific to the run files. While plan files have templates with possible parameter values, run files contain job descriptions with concrete parameter values.

Two alternative job definitions are provided by EnFuzion. The job statement defines a single job and its input values. The variable statement defines parameter values, which are then referenced to define a job. The following sections provide details on the job and variable statements.

The Job Statement

The **job** statement defines a job in a run file. A job consists of the task to execute and job parameters.

Jobs are specified as an optional task name followed by a list of parameters.

The syntax of the **job** statement is:

```
job \
    [ name <job_name> ] \
    [ task <task_name> ] \
    [ host <host_name> ] \
    [ node <node_id> ] \
    [ parameters <count> [ <par_name> <value> ] ... ]
```

Options are:

- **<job_name>**

  the name of the job. Default is "j<number>", where the number is uniquely assigned by the Dispatcher.

- **<task_name>**

  the name of the main job task. Default is "main".

- **<host_name>**

  the host name of the node to execute the job. If the node with this host name is not defined, an error message is returned.

- **<node_id>**

  the node id of the node to execute the job. If the node with this id is not defined, an error message is returned.

- **<count>**

  the number of job parameters. This is the number of <par_name>, < value> pairs that follow.

- **<par_name>**

  parameter name

- **<value>**

  parameter value. A string in double quotes "...".

Examples of job definitions:

```
job task main parameters 1 job_number "2";
job parameters 1 job_number "3";
job task dummytask parameters 1 job_number "4";
```

This example defines three jobs, each with one parameter called job_number. The first two jobs execute task main, the last job executes task dummytask.

The Variable Statement

The **variable** statement and its associated statements provide an alternative way to the **job** statement to describe jobs and their parameter values. The EnFuzion Generator uses this statement. The statement is useful, when parameter values are long strings and shared by many jobs, because job descriptions can be shortened significantly.

Possible parameter values are defined with statements **variable** and **indexcount**. These are followed by job definitions between the **jobs** and **endjobs** statements, which provide jobs and their input variable values.

The **variable** statement can specify either a list or a value. Lists are useful when the same parameter value is shared by many jobs. Values are useful when a parameter value is unique for a job. Counting for indexes and list elements starts with 0. Examples below illustrate the use of the **variable** statement.

Example:

```
variable job_number index 0 list "job1" "job2";
variable input_number index 1 list "input1" "input2";
indexcount 2
jobs
01 0 1
02 1 1
03 1 0
endjobs
```

This example defines three jobs, named 01, 02 and 03. For job 01, parameter job_number has value "job1" and parameter input_number has value "input2". For job 02, parameter values are "job2" and "input2", respectively. For job 03, parameter values are "job2" and "input1", respectively.

Example:

```
variable job_number index 0 list "job1" "job2" "job3";
variable param1 index 1 value;
```

```
variable input_number index 2 list "input1" "input2";
variable param2 index 3 value;
indexcount 4
jobs
01 0 "aaa" 1 "bbb"
02 1 "ccc" 1 "ddd"
03 2 "eee" 0 "fff"
04 1 "ggg" 0 "hhh"
endjobs
```

This example defines four jobs, named 01, 02, 03 and 04. For job 01, parameter job_number has value "job1", parameter param1 has value "aaa", parameter input_number has value "input2" and parameter param2 has value "bbb". For job 02, parameter values are "job2", "ccc", "input2" and "ddd", respectively. For job 03, parameter values are "job3", "eee", "input1" and "fff", respectively. For job 04, parameter values are "job2", "ggg", "input1" and "hhh", respectively.

Variables

Variables provide values for EnFuzion options and job parameters. By changing the values, default EnFuzion behavior can be modified and customized. Each variable is defined within a scope, which determines its visibility and how it is used. A scope can be a cluster, node, run, job, or context.

Variable Types

Variables come in two types, options and parameters.

Options

As options, variables control EnFuzion behavior. Options are predefined by EnFuzion, like values, for example. You can read an option to obtain its value or you can set it to change its value and thereby modify EnFuzion behavior. Some options are read only and cannot be set by the user. See the Section called *Options*.

Parameters

As parameters, variables may be predefined by EnFuzion or defined by the user. Parameters provide input values to user jobs, either in tasks or through the process environment. See the Section called *Parameters* below.

Scope

Variables are defined within a scope, which can be a cluster, node, run, job or context. Each scope has predefined options and system provided parameters. User defined parameters can be specified for any defined cluster, node, run, job or context.

Parameters for a particular job are a combination of parameters from different scopes. If the same parameter is specified in different scopes, then the value from the latest scope will override any previous value. Parameters for different tasks are combined from several scopes as follows:

Table 8-1. Available Parameters

Task Type	Parameters
rootstart, rootfinish	cluster parameters, run parameters, job parameters: single values, ENFOS, ENFOS_RELEASE, ENFMACHINE, ENFJOBNAME, ENFNODE
nodestart	cluster parameters, node parameters, run parameters, context parameters, job parameters: single values
main	cluster parameters, node parameters, run parameters, context parameters, job parameters

Only single value job parameters as described in the Section called *The Parameter Statement* are available for system tasks rootstart, rootfinish and nodestart. No other job parameters are available for the system tasks.

System parameters for **rootstart** and **rootfinish** provide values from the root machine.

Retrieving and Setting Values

EnFuzion provides several methods to retrieve and set variable values. All variables can be retrieved through the API. All parameter values can be retrieved by executing jobs, provided that they exist in the scope. All variables can be set through the API and by executing jobs. In addition, node variables can be set through node configuration file **enfuzion.options**, root variables can be set through root configuration file **root.options** and run variables can be set through run files.

Options

This section lists system defined options and provides their description. Options are grouped by: cluster, node, run, job and context.

Cluster Options

ENFMAIN_DIRECTORY: provides the cluster's main directory, used as a working directory by the Dispatcher. This option is read only.

ENFCLEANUP_LIMIT: sets the number of seconds the run directory is still available after the run completed. After the ENFCLEANUP_LIMIT expires, the run directory is deleted, if not deleted before by the user. By default, ENFCLEANUP_LIMIT is 7 days. ENFCLEANUP_LIMIT has no effect, if the run is executing in the single run mode.

ENFJOB_DAEMON_PORT: provides the port number of the job daemon. Job daemon is used by jobs on nodes to execute operations on the root. This option is read only.

*Options in **root.options***

Root options that can be specified in the **root.options** file are accessible as variables through the cluster object. Variable names are specified as option names in uppercase letters with the ENF prefix. For example, the completelogs option can be accessed through the ENFCOMPLETELOGS variable. Exceptions are off and on periods, which cannot be accessed as variables. See the Section called *Specifying Root Configuration Options* in Chapter 6 for a description of available options.

Node Options

ENFPROPERTIES: provides a list of node properties. By default, the list is empty.

Run Options

ENFPRIORITY_LEVEL: specifies a priority level for the run, which determines how nodes are allocated to runs. The default value is 50.

ENFPRIORITY_WEIGHT: specifies allocation weights for runs at the same level. The default value is 1.

ENFALLOCATION: provides current node allocation for the run. This option is read only.

ENFPERSISTENT: determines if the run is persistent or transient. If it is true, the run is persistent. If it is false, the run is transient. A persistent run must be terminated explicitly by the user. Jobs belonging to a persistent run are automatically deleted from the run after they are done. The default value is false.

ENFPREEMPTIVE: determines if the run is preemptive or non-preemptive. If it is true, the run is preemptive. If it is false, the run is non-preemptive. A preemptive run obtains required resources immediately after it is started. A non preemptive run waits until resources become available. The default value is false.

ENFEXECUTION_LIMIT: determines the time in seconds that the run is allowed to execute. If the limit is exceeded, the run is aborted. The default value is 0, which means no limit.

ENFJOB_EXECUTION_LIMIT: determines the time in seconds that each job in the run is allowed to execute. If the limit is exceeded, the job is aborted. The default value is 0, which means no limit.

ENFDATASTREAM_EXECUTION_LIMIT: determines the time in seconds that a result is expected from a datajob. If the limit is exceeded, the datajob is restarted on another machine. The default value is 0, which means no limit.

ENFMAX_JOB_COPIES: determines the maximum number of concurrent executions of the same job. It can be used to start multiple concurrent job executions, if nodes differ significantly in computer power. The default value is 1.

ENFPERMANENT: determines whether or not a permanent connection is maintained between jobs on nodes and the job daemon. If it is true, jobs maintain a permanent connection. If it is false, the connection is established on demand. The default value is false.

ENFDATAIN: contains the file name for datajob input. It has no default value.

ENFDATAOUT: contains the file name for datajob output. It has no default value.

ENFREQUIREMENTS: contains a list of run requirements. By default, the list is empty.

ENFLICENSES: contains a list of licenses, required by the run. By default, the list is empty.

ENFDATASTREAM_EVENTS: turns on and off datastream events. If false, no datastream events are generated. The default value is true.

ENFNODE_LIMIT: determines the maximum number of concurrently executing jobs. It is used to limit the number of nodes that the run uses. The default value is 1024, which means that all available nodes are used.

ENFFAIL_LIMIT: determines the number of successive jobs that can fail on a node. When that many successive jobs from the run fail on a single node, that node is not used any more for the run. The default value is 0, which means that this option has no effect and there is no limit on the number of jobs that can fail on the node.

ENFRESTART_LIMIT: determines the number of times that a job can be rescheduled in the case of an error. When this number is reached, the job is terminated with an error. The default value is 0, which means that this option has no effect and there is no limit on the number of times a job can be rescheduled. A job with an error is rescheduled for execution, if the **onerror repeat** or **onerror restart** option is present in the run file (see the Section called *Command **onerror*** for details).

ENFCPU_COUNT: determines the maximum number of CPUs that a job from the run is able to utilize. It is used to specify the optimal number of CPUs for one job. The actual number of CPUs available to the job might be smaller or larger, depending on other jobs being scheduled on the same node. ENFCPU_COUNT ensures that EnFuzion will not schedule any additional jobs on a node, where the sum of ENFCPU_COUNT of all the jobs executing on the node exceeds or is equal to the node **joblimit** value (for details on the node **joblimit** value, see the Section called *Requested Concurrent Jobs* in Chapter 7 and the Section called *Requested Concurrent Jobs* in Chapter 7). The default value for ENFCPU_COUNT is 1, which means that jobs use one CPU. The value of 0 means that jobs are able to utilize all available CPUs.

ENFDIRECTORY: provides the run subdirectory within the main cluster directory. This option is read only.

ENFNODE_USER: defines the account that the run owner wants to use on nodes to execute jobs. The actual account is determined with the **user.accounts** file. The account on each host is specified as "<user_name>[@<host_name>]". Accounts for several hosts can be specified in which case they are separated by a " ". If only an account is specified without a host, this is the default account for all hosts not explicitly mentioned.

ENFNODE_DIRECTORY: defines the job working directory on nodes. The directory on each host is specified as "<directory>[@<host_name>]". Directories for several hosts can be specified in which case they are separated by a space. If a directory name contains a space, it must be included in quotes. If only a directory is specified without a host, this is the default directory for all hosts not explicitly mentioned.

ENFNODE_NICE_PRIORITY: defines if the user jobs are executed at a background priority which utilizes idle cycles on the nodes. The priority on each host is specified as "on[@<host_name>]".

Priorities for several hosts can be specified in which case they are separated by a space. If only a priority is specified without a host, this is the default priority for all hosts not explicitly mentioned.

ENFNOTIFY_ADDRESS: defines an electronic address for notifications. An address is specified as "<user_name>[@<domain>]". Several addresses can be specified. Multiple addresses must be separated by spaces. If ENFNOTIFY_ADDRESS is not provided at the submission time, EnFuzion uses the run user identification as a default value. A mail server must be available to EnFuzion to send notifications. On most Linux/Unix systems, there is no need for any special configuration parameters. On Windows, a mail server must be specified as described in the Section called *Specifying Mail Server System* in Chapter 6.

ENFNOTIFY_CONDITION: defines conditions for an electronic notification. Whenever a condition is true, an e-mail is sent to all addresses in ENFNOTIFY_ADDRESS. A condition can be one of "abort", which denotes that a run was aborted, "start", which denotes a run start, "stop", which denotes a run stop, "approval", which denotes an approval point, and "done", which denotes that the run completed. By default, ENFNOTIFY_CONDITION has no value and no notifications are sent.

ENFAPPROVAL: defines a list of approval jobs. Multiple approval jobs can be specified. These jobs are executed first. When all the jobs in the approval list complete, either successfully or with a failure, the run priority level is set to 10. The run priority level is returned to its previous value after the user approves the run. Users can approve the run through the Eye. External tools can use the **run approve** API command to approve the run.

ENFPATH_SUBSTITUTE: defines a list of variables that will be transformed by EnFuzion at the execution time. Each variable contains a file path on the submit computer and will be transformed to a path on the node, following the instructions in the **paths** file. For details on the **paths** file, see the Section called *Specifying Path Correspondence* in Chapter 7. If this option is set, then ENFSUBMIT_PLATFORM must be specified as well.

ENFSUBMIT_PLATFORM: provides the operating system on the submit computer. It can be one of the following values: "windows", "linux" or "osx". This option must be set, if ENFPATH_SUBSTITUTE is used.

ENFACCESS: has a value "OK", if the caller has the permission to control run execution and access its files. Otherwise, it returns an error. This variable works also for completed runs.

Job Options

ENFJOB_REQUIREMENTS: contains a list of job requirements. By default, the list is empty.

Context Options

ENFCONTEXT_PROPERTIES: contains a list of context properties. By default, the list is empty.

Parameters

The following are system defined parameters. Parameters are grouped by: cluster, node, run, and job.

Cluster Parameters

ENFHOST: host that runs job daemon. Normally, the same as the root host.

ENFPORT: port for job daemon. Normally, the same as the variable ENFJOB_DAEMON_PORT.

Node Parameters

ENFNODE: unique node id.

ENFBIN: node directory with EnFuzion binaries.

ENFOS: node operating system.

ENFOS_RELEASE: node operating system release.

ENFMACHINE: node hardware platform.

ENFHOSTNAME: node host name.

Run Parameters

ENFRUN: run name.

ENFRUNID: run identifier.

ENFDIRECTORY: run directory on root.

ENFUSER: the run owner user ID.

ENFACCOUNT: user defined string for accounting purposes.

Job Parameters

ENFJOBNAME: unique job name.

ENFJOBCOUNT: contains the instance number of the job executing. This value is incremented every time the job is rescheduled, so that no two instances of the job have the same value.

ENFCWD: current job working directory on node.

Multiple Runs

EnFuzion is able to schedule jobs from multiple runs. Jobs are scheduled according to run priorities and attributes, such as preemption and persistence. These are described in more details below.

Priorities

EnFuzion schedules the execution of runs according to their priorities. Each run is allocated a certain number of nodes, based on its priority. The allocation is stored in the run variable ENFALLOCATION.

Whenever a node is ready to process the next job, a run with the highest priority, and one which is not using its allocated number of nodes, is selected to execute the job. If no such run is found, one of the remaining runs is selected to execute the job. The order in which runs are selected follows priorities first, then falls back to the order in which runs were submitted. If the queueing scheduling policy is turned on (see the Section called *Queueing Policy* in Chapter 6), then jobs are selected from the runs with the highest priority in the order in which runs were submitted.

Run priority consists of a run level and a weight. Run priority can be changed dynamically at runtime. EnFuzion adapts node allocation to reflect modified values.

Run Level

Runs at a higher level are executed before runs at a lower level. Jobs from lower priority runs wait to execute until all jobs from higher priority runs have executed. When jobs from higher level runs complete execution or are not utilizing all the nodes in the cluster, the execution of lower priority runs continues on idle nodes.

The default level for runs is 50. The run level is stored in the run variable ENFPRIORITY_LEVEL.

Run Weight

Runs at the same run level execute concurrently, but are allocated nodes proportionally to run weights. Runs with higher weights are allocated more nodes. For example, a run with weight 2 is provided with twice as many nodes for execution as a run with weight 1. If the queueing scheduling is turned on, then the weight has no effect.

The default weight for runs is 1. The run weight is stored in the run variable ENFPRIORITY_WEIGHT.

Preemption

Run preemption determines how runs start. Runs can be preemptive or non-preemptive.

When a run is started, it is allocated a certain number of nodes, according to its priority. A node is released when it requests a new job or a datajob to execute. Non preemptive runs postpone job execution until a node is released by another run. Preemptive runs do not wait for a node release, but immediately start terminating jobs from runs with lower priorities or runs that are using more than their allocated number of nodes. In general, preemptive runs start executing immediately at the expense of being disruptive to other runs.

By default, runs are non-preemptive. Run preemption status is stored in the run variable ENFPREEMPTIVE, which has a default value of false.

Persistence

Run persistence determines how runs are terminated. Runs can be persistent or transient.

Persistent runs are maintained by the Dispatcher in the ready state and never completed. They must be explicitly removed from the Dispatcher by the user by using the command **cluster remove run**

<run_id> or command **run <run_id> abort**. Persistent runs are useful for runs that process primarily streams of jobs, with jobs arriving at arbitrary times and possibly over a long time period.

Transient runs are completed by the Dispatcher and removed from the Dispatcher's internal tables as soon as all their jobs complete. Transient runs are useful for runs where all the jobs are known and specified at the beginning of the run and can be submitted before the run starts.

By default, runs are transient. Run persistence status is stored in the run variable ENFPERSISTENT, which has a default value of false.

Resource Management

Runs and the individual jobs that comprise them can specify requirements for job execution. These requirements are fulfilled by nodes through properties. EnFuzion executes jobs only on nodes that provide all the properties required by the job and its run.

Requirements

Requirements can be specified at the job or run level. Requirements at the run level are valid for all jobs in that run. Each individual job can have additional requirements.

Run requirements are specified in the run variable ENFREQUIREMENTS, which contains a list of requirements. By default, ENFREQUIREMENTS contains no requirements and is empty. Requirements can be added or deleted from ENFREQUIREMENTS through the EnFuzion API, run file or during job execution with the commands **set** and **unset**.

Job requirements are specified in the job variable ENFJOB_REQUIREMENTS, which contains a list of requirements for the job. By default, ENFJOB_REQUIREMENTS contains no requirements and is empty. Requirements can be added or deleted from ENFJOB_REQUIREMENTS through the API or during job execution with the commands **set** and **unset**.

Properties

Properties are specified for each node. A node property can be global for all runs or local to a particular run.

Node properties are specified in the node variable ENFPROPERTIES, which contains a list of properties. By default, ENFPROPERTIES contains no properties and is empty. Properties can be added or deleted from ENFPROPERTIES through the API, a node options file or during job execution with the commands **set** and **unset**.

Local run specific properties are set within a context . Local properties are specified in the context variable ENFCONTEXT_PROPERTIES, which contains a list of properties. By default, ENFCONTEXT_PROPERTIES contains no properties and is empty. Properties can be added or deleted from ENFCONTEXT_PROPERTIES through the API or during job execution with the commands **set** and **unset**.

Requirement Matching

Before a job is assigned to a node, it is verified that the global and local node properties, together, satisfy all job and run requirements. If the node does not satisfy all requirements, start of job execution is delayed until the next node becomes available.

See the Section called *Command **set*** and the Section called *Command **unset***, for setting parameters and requirements from a task and the Section called *Application Programming Interface* in Chapter 10, for setting parameters and requirements by means of the API.

Timeouts/Error Handling

EnFuzion provides several mechanisms to effect how job execution errors are handled. These are summarized in this section.

User Errors

With the **onerror** command, users can specify the handling of execution user errors. These can be caused by such things as missing files or user programs that return non-zero exit status. By default, a job with a user error fails. User errors can be either ignored with a successful job completion, or the job can be rescheduled for execution. See the Section called *Command **onerror***, for details on the **onerror** command.

If a job fails, EnFuzion executes an error handler. By default, the handler saves the current environment and copies it to the root machine are described in the Section called *Handling of Job Execution Errors* in Chapter 1. This default behavior can be changed by providing a user defined error handler in the **onerror** task (see the Section called ***onerror***).

Timeout for Run Execution

An execution limit can be specified for a run. If a run execution exceeds this limit, it is terminated.

By default, the run execution limit is infinite. The run execution limit is stored in the run variable ENFEXECUTION_LIMIT and contains the limit in seconds.

Timeout for Job Execution

An execution limit can be specified for jobs. The limit is independent of the run execution limit. If a job execution exceeds this limit, it is terminated with failure.

By default, the job execution limit is infinite. The job execution limit is stored in the run variable ENFJOB_EXECUTION_LIMIT and contains the limit in seconds. It is valid for all jobs in the run. This limit does not apply to individual datajobs.

Timeout for User Programs

An execution limit can be specified for user programs. While the ENFJOB_EXECUTION_LIMIT is valid for the entire job, the time limit for user programs specifies how long an individual user command within a job can execute. If a user program execution exceeds this limit, it is terminated with failure.

By default, the timeout for user programs is infinite. The timeout is specified with the task command **limit**, parameter complete. See the Section called *Command **limit***.

Multiple Job Executions

The same job can be executed concurrently on several nodes. This capability is useful when hosts differ widely in their computing speed. In this case, the slowest host can significantly delay run completion. With multiple execution , a job is concurrently started on several machines, provided that there are idle nodes and that the run uses less than its allocated nodes. The job completes when the first execution is completed. The remaining executions are terminated or ignored.

By default, only one copy of a job is executed concurrently. The maximum number of concurrent job executions is stored in the run variable ENFMAX_JOB_COPIES with a default value of 1.

Timeout for Datajob Execution

An execution limit can be specified for datajobs. The limit is independent of the job and run execution limit. If a datajob execution exceeds this limit, it is restarted on the next available node.

By default, the datajob execution limit is infinite. The datajob execution limit is stored in the run variable ENFDATASTREAM_EXECUTION_LIMIT and contains the limit in seconds. It is valid for all datajobs in the run.

Timeouts for Persistent User Programs

Various time limits can be specified for persistent user programs that execute datajobs. These values can limit the initialization time for user programs, the time for the initial connection with the persistent user program, the time to process one datajob, and the total time.

By default, all timeouts are infinite. Timeouts are specified with task command **limit**. See the Section called *Command **limit***.

Completed Run Directories

After a run is completed, its directory is kept until deleted by the user. To prevent accumulation of run directories, the Dispatcher automatically deletes obsolete run directories. The time limit for obsolete directories is specified by the cluster variable ENFCLEANUP_LIMIT. Its default value is 7 days or 604800 seconds. Obsolete directories are deleted only when the Dispatcher is executing in the multi run mode. ENFCLEANUP_LIMIT has no effect in the single run mode.

Datajobs

Datajobs provide higher throughput than regular jobs and significantly reduce the overhead associated with job execution.

In addition to delivering higher performance, datajobs work with persistent user applications. Persistent user applications are those that span multiple jobs. Persistent applications need to be initialized only once for all jobs and thus save often prohibitive initialization time for each individual job. Persistent applications also save time by removing the need for creating a new user process for each job.

Specifying Datajobs

Datastreams can be used in two modes, static mode and streaming mode. In static mode, all the input data is available at the beginning of the run. In streaming mode, new data can be added at any time during the run. Static and streaming modes cannot be mixed in one run. Each run contains either static or streaming datajobs.

Static Datajobs

For static datajobs, EnFuzion reads and writes directly from and to user files. There are no temporary files.

Static datajobs are handled through the API commands **usein** and **useout**.

```
run <run_name> usein datafile <filename>
```

Input data is taken from <filename>. The input file is not changed by EnFuzion. It is not copied, renamed or deleted. The ENFDATAIN variable is set to <filename> and is read only.

```
run <run_name> useout datafile <filename>
```

Output data is stored in <filename>. The output file is used by EnFuzion. There are no temporary files. The ENFDATAOUT variable is set to <filename> and is read only.

The input and output data files must be specified before datajobs start to execute.

Streaming Datajobs

For streaming datajobs, EnFuzion maintains temporary input and output files. User input and output data is stored in these files.

Temporary input and output files are limited in size to 100Mb. Once the size of a temporary file is greater than the limit, a new file is opened. When all the data is read from the file, the file is deleted.

Streaming datajobs are handled through the API commands **in**, **out**, **poll**, **movein**, **copyin** and **moveout**.

```
run <run_name> in data "<data>"
```

<data> is appended to the input file as the new datajob. It must be included in quotes.

```
run <run_name> out data
```

The next datajob result is returned and removed from the output file. Each line is prefixed with the ’data’ keyword. When no data is available "nodata" is returned. If the run has finished and no data is available, the command returns the string EOS.

```
run <run_name> poll data
```

Returns 0, if no results are available, returns > 0, otherwise.

```
run <run_name> movein datafile <filename>
```

The contents of <filename> are appended to input datajobs by renaming the file as the next temporary input file. The operation removes the original source file. This operation copies the content if the source and the destination file are on different file partitions. When files are on the same partition, the operation just renames the filename.

```
run <run_name> copyin datafile <filename>
```

The <filename> is appended to input datajobs by copying the file into the next temporary input file. The operation does not change the original source file. This operation always copies the file.

```
run <run_name> moveout datafile <filename>
```

All existing datajob output is stored in <filename>. Appropriate temporary output files are deleted.

Datajob Format

In datajob input files, different jobs are separated by the newline character. EnFuzion expects that datajob input does not contain quotes ’"’ or backslashes ’\’. If a quote, backslash, or a newline is required as part of datajob input, it must be escaped by preceding it with a backslash ’\’. EnFuzion removes these backslashes before a datajob is passed to the user program for execution.

Executing Datajobs

Datajobs are executed through the **server** task command. The command handles initiation and communication with a persistent user application. While the persistent application is executing datajobs, the job server acts as a link between the application and the Dispatcher. It requests new datajobs, sends them to the application for processing and returns the results to the Dispatcher.

See the Section called *Command **server***.

Chapter 9. Run Execution

User jobs are executed through the Dispatcher process on the EnFuzion root. This chapter describes the Dispatcher, the basic steps of run execution, which include run submission, execution monitoring and retrieval of run results, and accounting reports.

The Dispatcher executes on the EnFuzion root system. It can be used either to process a single run as a command line utility or multiple runs as a service on the network. If the Dispatcher is used as a service, submit computers are normally local user machines and are different from the EnFuzion root system.

When a run is submitted for execution, it is assigned an owner user ID by the Dispatcher. In most cases, this user assignment is done transparently to the EnFuzion user. An exception is the web browser interface, which requires an explicit user login. Otherwise, a generic **anonymous** user ID is assigned as the run owner. User identity can also be copied to other systems or accounts.

Each of the execution steps, which is either a run submission, monitoring or results retrieval, can be performed via a web browser or from a command line. The web browser capability is provided by an EnFuzion process on the root system, called the EnFuzion Eye. Transparently to the user, the Eye communicates with the Dispatcher and produces the pages for the web browser. The run execution steps are performed from a command line with programs **enfsub** and **enfcmd**. The programs can be executed directly by the user or used in scripts. The Dispatcher also provides a network based application programming interface, which can be used by custom programs to communicate directly with with the Dispatcher.

If the Dispatcher is terminated for any reason, or if the EnFuzion root system crashes unexpectedly, it is possible that some jobs might not have been completed, while other jobs may still be waiting for execution. Although the Dispatcher is able to automatically restart unfinished runs without jobs that already completed, EnFuzion provides a separate **enfpurge** utility. Enfpurge removes completed jobs from a run file, so that uncompleted jobs can be resubmitted.

The Dispatcher records accounting information, which provides details on how cluster resources are being used. The accounting information can be used to produce reports, which contain either run information, showing run use of node computers, or node information, showing node utilization.

The sections below provide details on the Dispatcher, run submission, including user assignment, execution monitoring, retrieval of results, the **enfpurge** program and on accounting reports.

The Dispatcher

The Dispatcher is the main program running on the EnFuzion root system, controlling job execution and other EnFuzion processes. It can be used to process a single run as a command line utility or multiple runs as a service on the network. The sections below provide details about Dispatcher parameters and details on using the Dispatcher for both single and multiple runs.

The Dispatcher Options

The Dispatcher command line is:

```
enfdispatcher   [ options ]   [ <run_file> ]
```

The Dispatcher reads its options and takes an optional run file. The optional run file is useful to provide the run description in a command line, when the Dispatcher is executed in a single run mode.

The Dispatcher command line options are:

- **-help**

  If this is the first option, then the Dispatcher prints out a help notice and exits. If it is not the first option, then **-help** has no effect.

- **-d**

  The Dispatcher is placed in a daemon mode . On Linux/Unix systems, the Dispatcher performs the following steps: forks twice, gets detached from the controlling terminal, becomes a session leader, and closes the standard file descriptors. On Windows, the Dispatcher calls itself with its original command line arguments, except for the "-d" argument, which is removed. The new process shares the same working directory, but is in a new process group, has a new console, which is not shown on the screen and does not inherit the handles. The original Dispatcher exits.

- **-m**

  The Dispatcher is executed in a multi run mode. By default, the Dispatcher is executed in a single run mode, where it executes one run either specified on a command line or a previously interrupted run and exits. In the multi run mode, the Dispatcher continuously processes runs until it is terminated by the administrator or by the system. The multi run mode is useful to provide EnFuzion as a network service.

- **-p <port_number>**

  This option changes the default port number of its network based application programming interface to **<port_number>**. By default, the Dispatcher uses port 10102. The application programming interface is described in the Section called *Application Programming Interface* in Chapter 10.

- **-r**

  This option recovers uncompleted runs from a previous Dispatcher. If the EnFuzion root system fails or the Dispatcher is terminated, then some of the runs might not be completed. If a new Dispatcher is restarted with the **-r** option in the same directory as the terminated Dispatcher, then it will reload the uncompleted runs and execute them.

- **-v**

  If this is the first parameter, then the Dispatcher prints out its version and exits. If it is not the first parameter, then **-v** has no effect.

- **-w <directory>**

The Dispatcher sets its working directory to the **<directory>** path. The working directory contains the Dispatcher log files and other working files.

This option is useful for safely setting the working directory, for example when the Dispatcher is executed using a scripting language or from a Java class.

- **<run_file>**

  This specifies the run file to process in a single run mode . Single run mode is suitable for executing the Dispatcher in scripts and from a command line. In single run mode, the Dispatcher takes a run file as input, automatically starts processing the jobs and exits after all the jobs complete. If all the jobs complete successfully, the Dispatcher returns 0 as its exit value. If some of the jobs fail, the Dispatcher returns 1 as its exit value.

  In single run mode, nodes are usually provided in the file enfuzion.nodes before the execution starts.

Most of the root options, described in the Section called *Specifying Root Configuration Options* in Chapter 6, can also be specified on the command line. The command line value takes precedence over the value in the **root.options** file. The root options that can be specified from the command line are:

- **-bind**, which determines if nodes can operate in the autonomous mode. See the Section called *Autonomous Node Operation* in Chapter 6 for details.
- **-cleanuplimit**, which specifies the period to delete the obsolete user directories. See the Section called *Deleting Obsolete User Directories* in Chapter 6 for details.
- **-commport**, which specifies the port to broadcast the root host and port on the local network. See the Section called *Port Number for Broadcasting the Address* in Chapter 6 for details.
- **-completelogs**, which turns on run specific events in the main cluster log. See the Section called *Complete Logs* in Chapter 6 for details.
- **-disconnect**, which specifies the period that either a root or a node machine waits for a heartbeat signal. See the Section called *Disconnect Period* in Chapter 6 for details.
- **-eyeport**, which specifies the Eye port number. See the Section called *Port Number for the Eye* in Chapter 6 for details.
- **-eyestart**, which specifies, if the Eye is automatically started by the Dispatcher. See the Section called *Starting the Eye* in Chapter 6 for details.
- **-eyeterminate**, which specifies, if the Eye is terminated by the Dispatcher. See the Section called *Terminating the Eye* in Chapter 6 for details.
- **-heartbeat**, which specifies the interval for heartbeat between the root and the node machines. See the Section called *Heartbeat Period* in Chapter 6 for details.
- **-httpport**, which specifies the port number for the HTTP based interface. See the Section called *Port Number for the HTTP Based Interface* in Chapter 6 for details.
- **-jobport**, which specifies the port number that is used by user jobs on EnFuzion nodes to execute services on the root. See the Section called *Port Number for Job Execution* in Chapter 6 for details.

- **-logsizelimit**, which limits the size of the Dispatcher log for log rotation. See the Section called *Maximum Dispatcher Log Size* in Chapter 6 for details.
- **-mailport**, which specifies port of the SMTP service host for electronic notification messages. See the Section called *Specifying Mail Service Port* in Chapter 6 for details.
- **-mailserver**, which specifies the SMTP server host for electronic notification messages. See the Section called *Specifying Mail Server System* in Chapter 6 for details.
- **-mailuser**, which specifies the sender for electronic notification messages. See the Section called *Specifying Mail Sender* in Chapter 6 for details.
- **-maxdatastream**, which specifies the maximum size for a datajob. See the Section called *Maximum Datastream Job Size* in Chapter 6 for details.
- **-maxstart**, which limits the number of concurrent node activations. See the Section called *Concurrent Node Activations* in Chapter 6 for details.
- **-multinodes**, which allows multiple nodes on a single computer. See the Section called *Multiple Remote Nodes from One Host* in Chapter 6 for details.
- **-noanonsubmit**, which denies run submission by users with the anonymous ID. See the Section called *Rejecting Anonymous Run Submission* in Chapter 6 for details.
- **-privileges**, which enforces user privileges. See the Section called *Enforcing Privileges* in Chapter 6 for details.
- **-protect**, which denies execution of user programs on the root system. See the Section called *Prevent Execution of User Programs on the EnFuzion Root System* in Chapter 6 for details.
- **-restart**, which specifies the node restart period. See the Section called *Node Restart Period* in Chapter 6 for details.
- **-rootport**, which specifies the port that is used by nodes to connect to the root when they are started independently. See the Section called *Port Number for Node Connections* in Chapter 6 for details.
- **-remoteaccess**, which denies remote access to the Dispatcher API port. See the Section called *Allowing Remote Access to the Dispatcher Interface* in Chapter 6 for details.
- **-resources**, which specifies how often nodes should report their resource usage. See the Section called *Minimum Time to Obtain Resource Information* in Chapter 6 for details.
- **-queue**, which turns on the queuing policy for scheduling. See the Section called *Queueing Policy* in Chapter 6 for details.
- **-startport**, which specifies the port that the enfnodestarter program uses to accept node requests during the node start sequence. See the Section called *Port Number for Node Starter Connections* in Chapter 6 for details.
- **-waitlimit**, which limits the time that nodes can operate in the autonomous mode. See the Section called *Wait Limit* in Chapter 6 for details.

Single and Multiple Run Execution

When the Dispatcher is used in single run mode, a run file must be supplied as an argument. The Dispatcher executes all of the jobs in the run and then exits. In this case, the submit computer and the root computer are the same. Input files and results are provided in the Dispatcher working directory on the submit computer.

When the Dispatcher executes multiple runs as specified with the **-m** command line option, the root computer is usually different from the submit computers. The Dispatcher is able to execute many runs concurrently, even from multiple users. Users submit their run files and their associated input files to the Dispatcher for execution. The submission is done through a web browser on the submit computer. This process is detailed in the Section called *Submission from a Web Browser*. EnFuzion also provides a command line program, which can be used to submit runs in scripts or from a command line. This process is detailed in the Section called *Submission from a Command Line*. Another option for submitting runs is through applications, using the EnFuzion API to communicate directly with the Dispatcher.

Handling of the Eye by the Dispatcher

The Dispatcher is configured to automatically handle the Eye, which provides a web based interface. The Eye is handled differently, depending on whether the Dispatcher is executed in a single run mode or in a multiple run mode.

If the Dispatcher is executed in a single run mode, then it starts the Eye at the beginning and terminates the Eye at the end of the run execution.

If the Dispatcher is executed in a multiple run mode, it starts the Eye at the beginning, but it does not terminate the Eye at the end. This allows remote users to access the result files even after the Dispatcher is terminated.

If the Eye process detects that another instance of the Eye is already executing on the system, using the same port to listen for requests, it terminates to prevent any conflicts.

Default behavior of the Eye can be changed by modifying the EnFuzion root configuration parameters, described in the Section called *Specifying Root Configuration Options* in Chapter 6.

Submitting a Run

User Assignment

When a run is submitted for execution, it is assigned an owner user ID.

If the run is submitted through a command line, this user identification and assignment are done transparently to the EnFuzion user. If the run is submitted through a web browser and the user is not logged in, a generic **anonymous** user is assigned as the run owner ID. If the user is logged in, its user ID is used.

The user performs a login by providing a user identification file. The file can be generated with the EnFuzion **enfcmd** command line utility. The EnFuzion user on the submit system can use a user

identification file to perform a login from a web browser. The file can also be copied to other systems or user accounts to identify the same user.

The following sections provide more details on the user identification file and the user assignment from a command line, a web browser or from a custom program that uses the EnFuzion API.

Identification from a Command Line

enfsub and **enfcmd** programs are used by users to communicate with the Dispatcher from a command line. They perform user identification transparently to the user. For most operations, there is no need for the user to issue any identification specific commands.

The **enfcmd** program is able to generate a user identification file, which can be used to log in from a web browser or to transfer user identity to another system.

> **Note:** The user identification file represents your EnFuzion user identity. It needs to protected from unauthorized access. Anyone that can read your user identification file, can use the file to log in to EnFuzion as yourself.

The following command generates a user identification file:

```
enfcmd identity
```

The file is named **<user>@<host_name>.enflogin**. **<user>** is the user account name on the submit system and **<host_name>** is the host name of the system. The file contains an encoded user identification string. The file can be copied to another system or user account to represent the same user. The user identification file needs to be generated only once.

The default user identification on the system can be changed by providing an ***.enflogin** file from a different user account or a different system. The file must be named **local.enflogin** and stored in the local directory or in the EnFuzion **config** directory on the submit host.

Identification from a Web Browser

A web browser is unable to obtain the local user account name and the host name. By default, users will be assigned a generic **anonymous** account, unless they explicitly log in to the Dispatcher.

Log in from a web browser is done by submitting a user identification file in the Eye Login page. A user identification file is generated with the **enfcmd** program as described previously in the Section called *Identification from a Command Line*.

Identification from a Custom Program

A program can connect to the Dispatcher API either as an **anonymous** user or it can provide a user identification string. Details of the connection protocol are described in the Section called *Establishing a Connection* in Chapter 10.

Submission from a Web Browser

The EnFuzion Eye program provides a web based interface for the Dispatcher. By default, the Eye is started automatically by the Dispatcher.

A user may interact with the Eye using a standard web browser, directed at the Eye port on the EnFuzion root host. The default port number for the Eye is 10101 on the EnFuzion root host. If the root host is the local system, the following URL connects to the Eye:

```
http://localhost:10101
```

Runs can be submitted on the Eye home page through the Submit A Run link. An alternative Submit link is also available in the header of all pages. Only the parametric execution runs, described in the Section called *Parametric Executions* in Chapter 8, can be submitted from a web browser. The command line runs and script runs must be submitted from the command line as described in the Section called *Submission from a Command Line*.

Run submission consists of several steps: a run file, input data files, and completion.

To submit a run, Click on the **Browse** button next to the **Run file** field, and select the run file. Clicking on the **Submit** button will copy the selected run file from your local system to the Dispatcher, and create a new run from it. If your run file was not formed correctly, an error message will be reported that adding the run failed. Otherwise, a page will be displayed, enabling you to select and upload optional data files. Select a file with the **Browse** button, and then click on the **Submit Data File** button to copy the file to the Dispatcher. You will see the data file added in the list below the submission form. Repeat this process for every input data file, until all input data files are submitted.

After all input files are submitted, select **Start Run Execution**, which will start run processing. The results of starting a run will then be displayed. If the run was successfully started, you can immediately view its state by clicking on the link providing the ID of the started run. This process is described with more detail in the Section called *Detailed Run Information Page* in Chapter 10.

> **Note:** Note that although EnFuzion allows you to specify a custom name for the run directory, custom directories are not supported by the Eye. The default directory, assigned by the Dispatcher, must be used.

If you are accessing the Eye via a proxy, it is possible that the proxy will not allow you to copy large data files. One solution is to bypass the proxy and connect to the Eye directly.

Details about the Eye are described in the Section called *Graphical Web Based Interface* in Chapter 10

Submission from a Command Line

EnFuzion provides a command line tool, called **enfsub**, which can be used to submit runs to the Dispatcher for processing.

The following sections provide details on submitting a run as a command line program, a script, or a parametric execution.

Submitting a Command Line Program

A command line program is submitted as follows:

```
enfsub  [ <enfsub_options> ]  <program>  [ <program_options> ]
```

The user program and its options are provided as parameters to the **enfsub** program. They can be preceded by <**enfsub_options**>, which are **enfsub** specific parameters.

An example:

```
enfsub sleep 30
```

Details about **enfsub** and its options are provided in the Section called *The Enfsub Program* in Chapter 10.

The following is a more complex command line example for Windows:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cmd /c copy input.txt output.file"
```

The following is the same command line example for Linux/Unix:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-\$ENFJOBNAME-\$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cp input.txt output.file"
```

Submitting a Script

Scripts are submitted similarly to command line programs:

```
enfsub  [ <enfsub_options> ]  <script>  [ <script_options> ]
```

The script and its options are provided as parameters to the **enfsub** program. They can be preceded by <**enfsub_options**>, which are **enfsub** specific parameters.

An example:

```
enfsub myscript.sh
```

This example assumes that **myscript.sh** is already available on all the nodes and that it is included in the execution path. If that is not the case, then the following example copies the script to the node and executes it from the current directory:

```
enfsub -i myscript.sh ./myscript.sh
```

The following is a Windows script that has the same effect as the command line example in the previous section:

```
@echo off
rem ENF -i script.bat
rem ENF -n sample -a myaccount
rem ENF -i input.txt
rem ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
rem ENF -rd -count 2 -e user@domain.com -m d

copy input.txt output.file
```

The script is submitted with:

```
enfsub script.bat
```

The following is a Linux/Unix script that has the same effect as the command line example above:

```
#!/bin/sh
#ENF -i script.sh
#ENF -n sample -a myaccount
#ENF -i input.txt
#ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
#ENF -rd -count 2 -e user@domain.com -m d

cp input.txt output.file
```

The script is submitted with:

```
enfsub ./script.sh
```

Details about **enfsub** and its options are provided in the Section called *The Enfsub Program* in Chapter 10.

Submitting a Parametric Execution

On Windows, runs can be submitted simply with a double click on the run file. The EnFuzion installation registers the **enfsub** program with Windows, so that **enfsub** is invoked for files that end with the **.run** suffix. The **enfsub** program also identifies and copies required input files, so these files are handled automatically.

From the command line or on Linux/Unix, runs are submitted for execution with the **enfsub** program:

```
enfsub  [ <enfsub_options> ]  [ -run ]  <run_file>  [ <input_files> ]
```

<run_file> and its **<input_files>** are submitted for execution to the Dispatcher.

The **-run** option can be omitted, if the **<run_file>** ends with the **.run** suffix. The **enfsub** program automatically detects input files, so the **<input_files>** arguments can be omitted from the command line.

Details about **enfsub** are described in the Section called *The Enfsub Program* in Chapter 10.

Submission from a Custom Program

The Dispatcher provides two different interfaces, which can be used by other applications to interact with the Dispatcher. The HTTP based interface (see the Section called *HTTP Based Application Programming Interface* in Chapter 10) is suitable primarily for job submission and retrieval of results. The EnFuzion API interface provides a comprehensive set of commands to monitor and control the Dispatcher. This section explains how to use these interfaces to submit jobs for execution.

Submission with the HTTP Based Interface

The HTTP based interface accepts a set of HTTP requests. Several HTTP requests are needed to submit a run. The process of submitting a run is as follows:

- create a new run with the **POST newrun** command:

  ```
  POST /cgi/newrun?runname=<run_name>&username=<user_name>&account=<account>
  ```

  Arguments to the **newrun** are optional. The request returns an ID for the new run. This ID is used in the subsequent requests to identify the run.

- upload input files to the EnFuzion Dispatcher with the **PUT** request:

  ```
  PUT <run-ID>/<file_name>
  ```

  Arguments are mandatory. The body of the request must contain file content. <file_name> is the target file path and name in the run directory on the EnFuzion root computer.

- submit the run for execution with the **POST startup**:

  ```
  POST /cgi/startrun?runid=<run-ID>&runfile=<file_name>
  ```

  Arguments are mandatory. <runfile> is a run file that must exist in the run directory on the EnFuzion root computer. It can be copied to the directory in a previous step.

Details about the HTTP based interface are provided in the Section called *HTTP Based Application Programming Interface* in Chapter 10.

Submission with the EnFuzion API

The Dispatcher provides a set of socket based commands, called API commands, which can be used by any program to monitor and control the Dispatcher. These commands can be used to submit runs to the Dispatcher for processing.

The API commands assume that the run file is available on the root computer and that the user is able to copy files to the run directory on the root computer.

The steps below detail how a program submits a run using the API commands:

- Connect to the Dispatcher API port number.
- Send the string "director" to the Dispatcher. The Dispatcher should return string "OK". This command connects to the Dispatcher under the anonymous user. See the Section called *Establishing a Connection* in Chapter 10 for details on how to log in under a different user ID.
- Create the run by submitting the run file to the Dispatcher using the following API command:

  ```
  cluster add run file <run_file>
  ```

  <run_file> is the run file name, relative to the main Dispatcher directory. The command creates a new run and returns its identification number.
- Copy the run input files to the run directory on the EnFuzion root, using external commands, provided by your operating environment. Alternatively, the HTTP based interface can be used for this step. The EnFuzion API currently does not support this functionality. The run directory is named **run-<run_id>**, where <run_id> is the run identification number, returned in the step above.
- Start run execution using the following API command:

  ```
  run <run_id> start
  ```

  <run_id> is the run identification number, returned in the first step above.

Details about the API commands are described in the Section called *Application Programming Interface* in Chapter 10

Resubmitting Unfinished Jobs

If the Dispatcher is terminated for any reason, then some jobs might not be completed. If the Dispatcher is restarted with the **-r** argument, it will automatically reload all unfinished jobs from a previous Dispatcher instance and continue their execution. However, sometimes it is useful to have a manual control over the restart process. This manual control is provided by the **enfpurge** utility.

The **enfpurge** utility allows you to create a run file which contains only jobs that have not been successfully executed. The utility is described in the next section.

Enfpurge

During execution, the Dispatcher creates a run log file that records which jobs have been successfully completed. The enfpurge utility takes a run file and its log and produces on standard output a run file consisting only of jobs that have not been completed. The output run file can be submitted to the Dispatcher to execute the remaining jobs.

The syntax of **enfpurge** is:

```
enfpurge <input_run> <log_file> <run_ID> > <output_run>
```

The following command line takes the run file **first.run** and the log of run 0086800000 in **enfuzion-run.log** and generate a new run file named **next.run**:

```
enfpurge first.run enfuzion-run.log 0086800000 > next.run
```

Monitoring Execution

EnFuzion provides several methods to monitor job execution. These include extensive Dispatcher logs, web based monitoring, command line monitoring and monitoring from custom programs. Details are provided in the sections below.

Dispatcher Logs

EnFuzion produces extensive logs which provide detailed information on EnFuzion operation. Logs record important events about all major objects in EnFuzion: the cluster, nodes, runs, jobs and datastreams.

The main log is called **enfuzion.log**. It is created in the main cluster directory. By default, **enfuzion.log** contains all events. Run specific events can be turned off to reduce overhead and increase performance.

Each run has its own log, called **enfuzion-run.log**. The log is created in the run directory. Run logs contain run, job, and datastream events. Datastream events can be turned off to reduce overhead and increase performance.

The enfuzion.log File

During execution, the Dispatcher produces a log, describing major execution events. The log is saved to the file **enfuzion.log**. Whenever a log grows too large, it is renamed to **enfuzion-%d.log**, where %d is the smallest integer with a nonexistent file.

Size of Dispatchers log is controlled through root option logsizelimit. The default size of the logsizelimit root option is 10 MB. Root options are described in the Section called *Specifying Root Configuration Options* in Chapter 6.

When a new Dispatcher is started, existing files **enfuzion-%d.log** and **enfuzion.log** are renamed to **enfuzion-%08x-%d.log** and **enfuzion-%08X.log**. Where %08x stands for a unique suffix. This preserves all old Dispatcher logs.

The Dispatcher log records all cluster events and execution statistics. Run events, job events and datajob events are recorded in the run log as well. The run log is created in the home directory of the run.

Execution statistics is provided when the root goes to a non-active state, when a node goes down, and when the run is done. Reports begin with the event report "======= execution report: =======".

Description of Log Events

Cluster Log Events

```
<time> <event_id> cluster <cluster_name> create port <port_number>
<time> <event_id> cluster <cluster_name> cleanup <statistics>
```

```
<time> <event_id> cluster <cluster_name> start
<time> <event_id> cluster <cluster_name> down <statistics>
<time> <event_id> cluster <cluster_name> message <text>
<time> <event_id> cluster <cluster_name> report <text>
<time> <event_id> cluster <cluster_name> add run <run_name>
<time> <event_id> cluster <cluster_name> remove run <run_name>
<time> <event_id> cluster <cluster_name> add node \
            <node_name> on host <host_name>
<time> <event_id> cluster <cluster_name> remove node <node_name>
<time> <event_id> cluster <cluster_name> set <variable_name> <value>
<time> <event_id> cluster <cluster_name> unset <variable_name>
```

Node Log Events

```
<time> <event_id> node <node_name> created on host  <host_name>
<time> <event_id> node <node_name> start
<time> <event_id> node <node_name> active
<time> <event_id> node <node_name> terminate
<time> <event_id> node <node_name> down <statistics>
<time> <event_id> node <node_name> idle
<time> <event_id> node <node_name> executing
<time> <event_id> node <node_name> busy <message>
<time> <event_id> node <node_name> removed
<time> <event_id> node <node_name> message <text>
<time> <event_id> node <node_name> report <text>
<time> <event_id> node <node_name> set <variable_name> <value>
<time> <event_id> node <node_name> unset <variable_name>
```

Run Log Events

```
<time> <event_id> run <run_name> create <data>
<time> <event_id> run <run_name> cleanup <statistics>
<time> <event_id> run <run_name> start
<time> <event_id> run <run_name> stop
<time> <event_id> run <run_name> continue
<time> <event_id> run <run_name> abort
<time> <event_id> run <run_name> stage <run_stage>
<time> <event_id> run <run_name> done
<time> <event_id> run <run_name> fail
<time> <event_id> run <run_name> add job <job_name>
<time> <event_id> run <run_name> add task <task_name>
<time> <event_id> run <run_name> change task <task_name>
<time> <event_id> run <run_name> datain <filename>
<time> <event_id> run <run_name> message <text>
<time> <event_id> run <run_name> report <text>
<time> <event_id> run <run_name> set <variable_name> <value>
<time> <event_id> run <run_name> unset <variable_name>
```

Job Log Events

```
<time> <event_id> job <run_name> <job_name> start <node> <host>
<time> <event_id> job <run_name> <job_name> reschedule <node> <host>
<time> <event_id> job <run_name> <job_name> restore <node> <host>
<time> <event_id> job <run_name> <job_name> resources \
```

```
                exectime <sec> usercpu <msec> kernelcpu <msec> \
                memory <Kb> pagefaults <int>
    <time> <event_id> job <run_name> <job_name> done <node> <host>
    <time> <event_id> job <run_name> <job_name> ignore <node> <host>
    <time> <event_id> job <run_name> <job_name> fail <node> <host>
    <time> <event_id> job <run_name> <job_name> abort
    <time> <event_id> job <run_name> <job_name> message <text>
    <time> <event_id> job <run_name> <job_name> set <variable_name> <value>
    <time> <event_id> job <run_name> <job_name> unset <variable_name>
```

Datastream Log Events

```
    <time> <event_id> datastream <run_name> <job_name> start <node> <host>
    <time> <event_id> datastream <run_name> <job_name> done <node> <host_name>
    <time> <event_id> datastream <run_name> <job_name> reschedule <node> <host>
```

Monitoring from a Web Browser

EnFuzion monitoring is available by connecting a standard web browser to the Eye program on the EnFuzion root host. The default port for the Eye is 10101.

The Eye provides several different monitoring pages. There is the main page for the overall EnFuzion cluster, a page with summary information for all the nodes, a page with details about each node, a page with summary information for all runs and a page with details about each run.

Cluster Page

The main EnFuzion monitoring page can be reached through the Cluster Monitoring link on the Eye home page. An alternative Cluster link is available in the header of all pages. This page gives basic information about the EnFuzion cluster status, uptime, nodes and runs. It also contains the log and any messages from the Dispatcher.

Node List Page

The node summary page can be reached through the Nodes link on the main monitoring, Cluster page. An alternative Nodes link is available in the header of all pages. This page gives each node’s name, status, uptime, distribution of time, and summary of job execution.

Single Node Page

A detailed node page can be reached through the link in the Name field of the node summary, Node List page. This page gives node details, including the node name, host, user, operating system, node start parameters, status, uptime, time distribution, and job execution.

Run List Page

This run summary page can be reached through the Runs link on the main monitoring, Cluster page. An alternative Runs link is available in the header of all pages. This page gives each run's name, status, uptime, scheduling parameters, and summary of job execution.

Single Run Page

A detailed run page can be reached through the link in the ID field of the run summary, Run List page. This page gives run details, including the run name, scheduling parameters, status, uptime, job execution, initialized nodes, results, log, and control.

Monitoring from a Command Line

EnFuzion provides a command line tool enfcmd, which can be used to monitor the Dispatcher. The **enfcmd** command connects to the Dispatcher API port, which is reported by the Dispatcher in its main log file, called **enfuzion.log**.

The main **enfcmd** option for monitoring is the **show** option. It provides detailed information about the entire EnFuzion cluster, or its individual components.

The syntax for the **show** option is:

```
show [ detailed ] [ ( cluster | node <node_id> | run <run_id> ) ]
```

- **detailed**

  an optional parameter to the show option. **show detailed** provides significantly more details than **show**. **detailed** can be added to any show parameter.

- **show**

  provides information about the cluster, its nodes and runs.

- **show cluster**

  provides information about the cluster.

- **show node <node_name>**

  provides information about the named node.

- **show run <run_id>**

  provides information about the named run.

Monitoring from a Custom Program

The Dispatcher provides a set of socket based commands, called API commands, which can be used by any program to monitor and control the Dispatcher.

A custom program connects to the Dispatcher as follows:

- Connects to the Dispatcher API port number. The port is provided in the main log, called **enfuzion.log**.
- Sends the string "director" to the Dispatcher. The Dispatcher should return the string "OK".

The Dispatcher is now ready to accept commands from the custom program. The monitoring commands are:

- **cluster get status**

  Returns the cluster status, which can be Down or Running.

- **cluster get statistics**

  Returns statistics about cluster execution.

- **cluster get nodes**

  Returns a list of nodes.

- **cluster get runs**

  Returns a list of runs.

- **node <node_id> get status**

  Returns the node status, which can be Executing, Idle, Busy, or Down.

- **node <node_id> get statistics**

  Returns statistics about node execution.

- **run <run_id> get status**

  Returns the run status, which can be Created, Started, Done, Failed, or Stopped.

- **run <run_id> get statistics**

Returns statistics about run execution.

Retrieving Results

Results from run execution are stored as files in the run directory, named **run-<run_id>**. The directory is located on the EnFuzion root system in the main Dispatcher directory.

EnFuzion includes a range of tools that simplify retrieval of results. Result files can be retrieved directly using system provided utilities, from a web browser, from a command line, or from a custom program.

Retrieving Files on the EnFuzion Root System

Files in the run directory on the EnFuzion root system can be retrieved using system provided utilities and browsers. These include the **copy** command on Windows and the **cp** command on Linux/Unix. If the main Dispatcher directory is exported for network access, then files can be retrieved from a remote system across the network.

After the run completes, and all relevant files have been retrieved from the directory, the run directory can be deleted by the user. If the directory is not deleted by the user, it will be deleted as specified in the **cleanuplimit** root configuration option.

Retrieval with a Web Browser

Results can be retrieved by connecting a standard web browser to the Eye program on the EnFuzion root host. The default port for the Eye is 10101.

The main link to retrieve results is the Check Run Results link on the Eye home page. An alternative Results link is available in the header of all pages.

The main results page provides a list of run directories, their status and the time of the last modification of the directory contents. The contents of these directories and the files they contain can be listed and downloaded in a way similar to browsing a file system with a file manager.

Retrieval from a Command Line

EnFuzion provides the command line tool enfsub, which is able to retrieve the results in addition to submitting a run. Enfsub can also attach to an existing run. Enfsub implements a range of options, so it can be used in a range of scenarios. Some common examples of enfsub usage are shown below.

Submit a run and exit immediately:

```
enfsub sample.run
```

Submit a run and wait for the run to complete:

```
enfsub -wait sample.run
```

Submit a run, wait for the run to complete, and copy its directory to a subdirectory on the local host:

```
enfsub -rd sample.run
```

Submit a run and fetch output files as they are begin created:

```
enfsub -fetch -pd 1 sample.run
```

Attach to an existing run, wait for the run to complete, and copy its directory to the local current working directory:

```
enfsub -attach -results sample.run
```

Additional enfsub options are described in the Section called *The Enfsub Program* in Chapter 10.

Retrieval with a Custom Program

External applications can retrieve results by using the HTTP based interface. Direct retrieval of files, using the EnFuzion API interface, is not supported.

The HTTP interface can be used to retrieve all the results after the run completes or to retrieve the results incrementally as jobs are still executing.

Results can be retrieved at the end of a run as follows:

- check that the run completed with the **POST runcompleted** command:

  ```
  POST /cgi/runcompleted?runid=<run-ID>
  ```

  The argument is mandatory. When this request returns 1, move to the next step. Otherwise, try again later.

- get a list of all run files with the **POST getallfiles** command:

  ```
  POST /cgi/getallfiles?runid=<run-ID>
  ```

  The argument is mandatory. This request returns a list of all files in the run directory.

- download all the files from the list using the **GET** request;

  ```
  GET <run-ID>/<file_name>
  ```

  Arguments are mandatory. The body of the response contains file content. <file_name> is the source file path and name in the run directory on the EnFuzion root computer.

The process of incremental file retrieval is as follows:

- get a list of new files with the **POST getnewfiles** command:

  ```
  POST /cgi/getnewfiles?runid=<run-ID>
  ```

  The argument is mandatory. This request returns a list of new files in the run directory.

- download all the files from the list using the **GET** request;

  ```
  GET <run-ID>/<file_name>
  ```

  Arguments are mandatory. The body of the response contains file content. <file_name> is the source file path and name in the run directory on the EnFuzion root computer.

- reset the copy mark with the **POST setcopymark** command;

  ```
  POST /cgi/setcopymark?runid=<run-ID>
  ```

  The argument is mandatory. This request sets the new mark for incremental file copying.

- repeat the steps until the list is empty and the run completes. Run completion is tested with the **POST runcompleted** command:

  ```
  POST /cgi/runcompleted?runid=<run-ID>
  ```

  The argument is mandatory.

Details on the HTTP based interface can be found in the Section called *HTTP Based Application Programming Interface* in Chapter 10.

Producing Accounting Reports

EnFuzion implements accounting reports, which show how cluster resources are being used. Reports can contain either run information, which shows run use of node computers, or node information, which show node utilization.

Reports are available at several levels of granularity. Hourly reports are maintained for the last two days, daily reports are maintained for the last two months and monthly reports are maintained indefinitely. Specific reports can be generated by grouping or selecting columns.

Accounting reports can be produced from a web browser or with a command line utility as described in the following sections.

Reports from a Web Browser

Accounting reports can be accessed by connecting a standard web browser to the Eye program on the EnFuzion root host. The default port for the Eye is 10101. The main link to retrieve accounting reports is the Accounting link on the Eye home page. An alternative Accounting link is provided in the header of all pages. The accounting page lists available run and node activity reports. The format of these pages can be changed through the button labeled Change Report Layout. More details on the Accounting pages are available at the Section called *Accounting Page* in Chapter 10.

Reports from a Command Line

EnFuzion provides the **enfreport** program to generate reports from a command line. **enfreport** allows EnFuzion users to create customized activity reports. It generates tabular reports on node and run activity with columns selected by the user. The output format may be either plain text, HTML or a CSV (Comma Separated Value) file.

enfreport is described in detail in the next section.

The enfreport Program

Enfreport has the following options:

```
enfreport \
    [ -type runs | nodes ] \
    [ -format text | csv | html ] \
    [ -root <working_directory> ] \
    [ -time <time_specification> ] \
    [ -columns <column_specification> ] \
    [ -group <name> ]
```

- **-type runs | nodes**

  This option selects the report type, which is either a run or a node report. The default value is **runs**. The report type determines values shown in the report. Run reports show node use by runs, and node reports show node utilization.

- **-format text | csv | html**

  This option selects the report output format, which is either text, HTML or CSV, comma separated values. The default value is text.

- **-root <working_directory>**

  This option specifies the directory with the accounting information. The directory can be either the Dispatcher working directory, which contains the **enfinfo/acct** subdirectory, or the actual directory with the accounting files, such as **/<path>/enfinfo/acct**. The default option value is the **enfreport** working directory. Normally, the value of the root option is the Dispatcher working directory, where accounting information is being stored automatically.

- **-time <time_specification>**

  This option selects the report time interval, which can be an hour, a day or a month. Hourly reports are available for the current and the previous calendar day, daily reports are available for the current and the previous calendar month and monthly accounts are kept indefinitely.

  <time_specification> is one of the following:

  - **-time H[[[<year>-]<month>-]<day>-]<hour>**

produces an hourly report. If the year, the month or the day are omitted, the current date values are used.

- **-time D[[<year>-]<month>-]<day>**

  produces a daily report. If the year or the month are omitted, the current date values are used.

- **-time M[<year>-]<month>**

  produces a monthly report. If the year is omitted, the current date values are used.

A report for the period between 12:00 and 13:00 of the current day is specified as "H12", whereas the same period on 30th of March 2001 should be specified as H2001-03-30-12. Similarly, a report for April 1st of this year would be denoted by time specification string "D04-01" and a report for the whole month of April would be specified as "M4".

- **-columns <column_specification>**

  This option selects the columns shown in the report. By default, all columns are shown.

  The <column_specification> string is a comma-delimited list of column definitions. Since spaces may be part of column names, make sure to include the string in quotes on the command line in order have it interpreted as a single command line argument. <column_specification> is one of the following:

  - **<column_name>**

    include the <column_name> in the report table;

  - **!<column_name>**

    exclude the <column_name> from the report table. If the "!" is the first item without <column_name>, then all columns are excluded.

  - **<column_name>=<value>**

    include only rows where the value in the <column_name> matches <value>.

  The list of available column names may be listed with the following commands:

  ```
  enfreport -type runs -help columns
  enfreport -type nodes -help columns
  ```

  Enfreport prints the following column names that may be used in column definitions:

```
* Host Name
* Node ID
  Uptime
  Downtime
  Executing Time
  Idle Time
  Busy Time
  Jobs Done
  Jobs Started
  Avg Job Length
  Max Job Length
```

The columns marked with an asterisk "*" are key columns. If one or more key columns are excluded from the report, rows with same values of the remaining key columns are combined to one row.

- **-group <name>**

  selects only rows with users from this group.

Below are a few examples of accounting reports. They assume the time between 12:00 and 13:00 today.

Show names and IDs of all nodes - exclude all columns and include only Name and ID columns:

```
enfreport -time H12 -type nodes -columns '!,Name,ID'
```

Show a report with a single row containing the number of done and started jobs for all runs:

```
enfreport -time H12 -type runs -columns '!,Jobs Done,Jobs Started'
```

Again, but this time only for runs owned by bob@qa.corporation.com:

```
enfreport -time H12 -type runs \
      -columns '!,Jobs Done,Jobs Started,User=bob@qa.corporation.com'
```

Show all runs owned by bob@qa.corporation.com, but without the ID, User and Account columns:

```
enfreport -time H12 -type runs \
      -columns '!ID,!User,!Account,User=bob@qa.corporation.com'
```

Show all runs owned by users in group QA:

```
enfreport -time H12 -type runs -group QA
```

Chapter 10. Interfacing with the Dispatcher

Users can interface with the Dispatcher by using the EnFuzion Eye and a web browser, or through the command line utilities enfsub and enfcmd. Custom programs can also communicate with the Dispatcher using its network based, programming interface. Chapter 9 describes how users and custom programs can accomplish most common tasks.

This chapter provides details about the Eye program, the command line program the enfsub and the enfcmd, and the Dispatcher programming interface.

Graphical Web Based Interface

The Eye program provides your EnFuzion cluster with an intuitive, web-based interface. It establishes a connection to the EnFuzion Dispatcher and displays information about a running cluster. The Eye uses a set of web pages, so that the user can interact with EnFuzion using a graphical web browser.

The Eye allows the user to monitor the state of the cluster, nodes and runs that EnFuzion uses. Furthermore, the Eye allows the inspection of cluster and run logs. Using a web browser, it can be used to browse and retrieve run results and to submit new runs and related data files.

The Eye runs as a separate program, interfering as little with the actual EnFuzion cluster as possible. If you encounter a problem while using the Eye, your cluster should continue functioning normally.

The Eye

The Eye is started by executing the **enfeye** executable residing on the root machine in the same location as the EnFuzion Dispatcher.

The Eye is normally started automatically by the Dispatcher, as described in the Section called *Handling of the Eye by the Dispatcher* in Chapter 9, so there is no need to change any of the configuration defaults to use the Eye.

The Eye can also be started manually from a command line, or its default configuration can be changed. The Eye command line options are described in the Section called *Eye* in Chapter 11. The Eye configuration options are described in the Section called *Specifying Root Configuration Options* in Chapter 6.

Using the Eye

Once the Eye is started as described above, you can use your web browser to connect to it. The Eye uses only plain HTML, conforming to the W3C HTML 4.01 DTD, and cascading style sheets to construct its web pages. The Eye works best with Internet Explorer 5 or higher, Mozilla 1.0 or higher and Netscape 6 or higher. Cookies must be enabled in your browser, in order for the Eye to function properly.

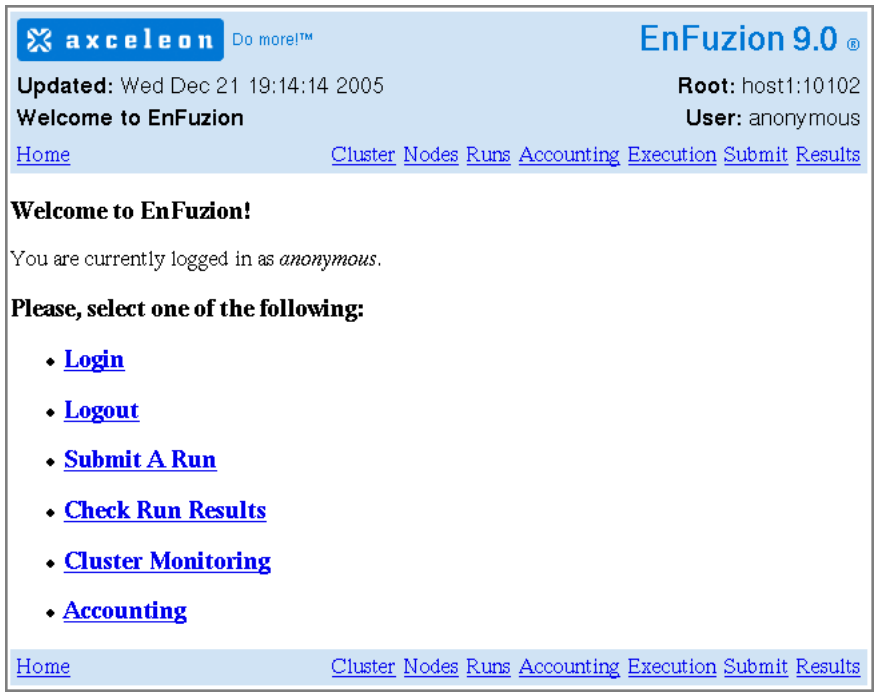
Your web browser needs to be directed to the system where the Eye is executing and to the port that the Eye is listening on. The default port number is 10101. Using default values, you can connect to the Eye with the following link:

```
http://<root_host>:10101
```

The Eye port number can be changed as described in the Section called *Port Number for the Eye* in Chapter 6.

Upon establishing a connection, you arrive at the Eye home page (see Figure 10-1).

Figure 10-1. The Eye Home Page



The header, which is common to most of the Eye pages, presents a short descriptive title of the page on the left-hand side, just below the Axceleon logo. The left hand side displays the time when the information used in creating the page was last updated and the title of the page being viewed. On the right hand side, the hostname and port where the Dispatcher is listening and the user that the Eye is currently logged in as are displayed.

The navigation bar in the header provides quick access to the most common activities. On the left, the option "Home" should always bring you to the home page that you are currently observing. The other options, which are described in the sections that follow, take you to the pages listed below:

- Cluster: Cluster State page
- Nodes: Node List page
- Runs: Run List page
- Accounting: Accounting Reports page
- Execution: Executing Job List page

- Submit: Run Submission page
- Results: Run Results page

This navigation bar is also replicated in the footer of each page.

The Eye home page offers you a choice of activities:

- The "Login" link lets you submit new login information.
- The "Logout" link gives you an "anonymous" user ID.
- The "Submit A Run" link allows you to submit a run
- The "Check Run Results" link presents you with a list of directories containing run results. Their contents may then be inspected and retrieved.
- The "Cluster Monitoring" link takes you to a set of pages that show information on the overall cluster state, as well as the runs and nodes used by the cluster.
- The "Accounting" link takes you to the page that lets you generate and view reports of EnFuzion activity.

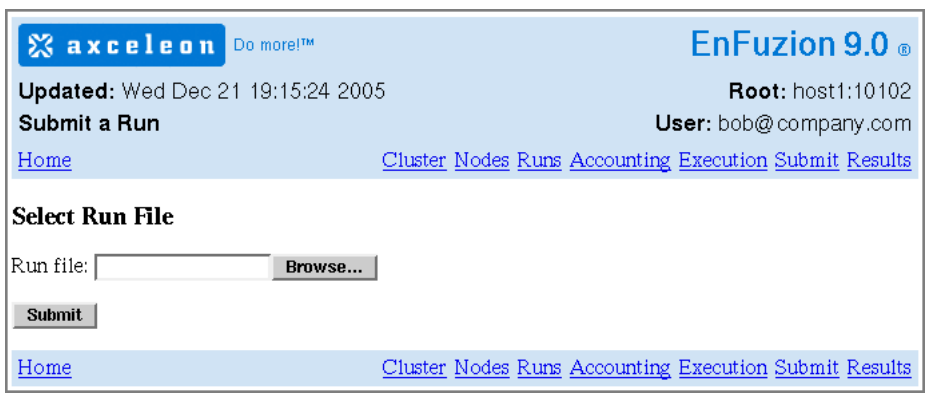
Most of the information in the Eye is presented in tables. When appropriate, the table contents may be sorted by column, in either ascending or descending order. If the column header is a hyper link, simply click on it to sort the table by that column.

A table that consists of more than a hundred rows is broken into pages of hundred rows each. In this case a page index appears above the table, displaying the current page number and links that allow for navigating the pages.

Submitting a Run

Runs can be submitted through the Run Submission page, which can be reached via the Eye home page or through the Submit link, available in the header menu. The Run Submission page is shown in Figure 10-2.

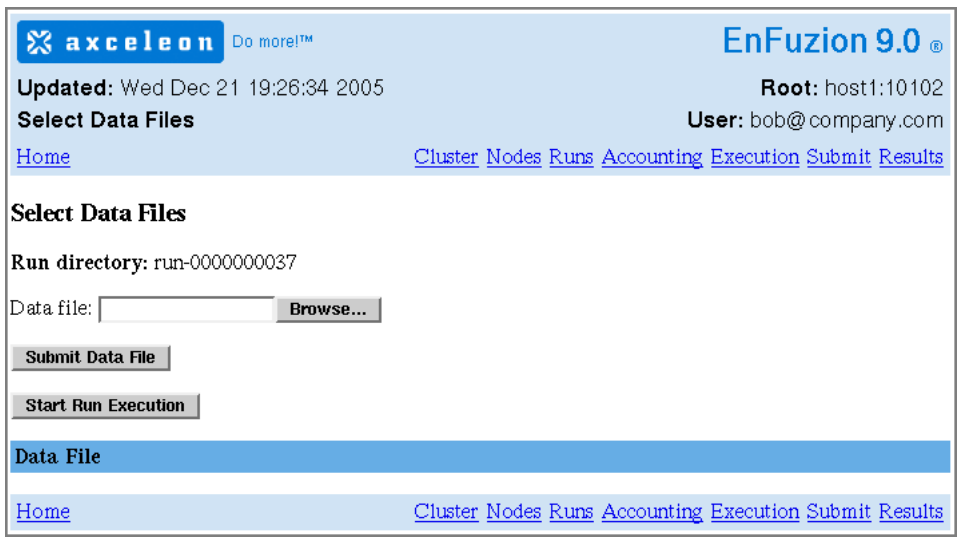
Figure 10-2. The Run Submission Page



When submitting a run, you first need to upload the run file to the Dispatcher. Click on the **Browse** button near the **Run file** field, and select your run file.

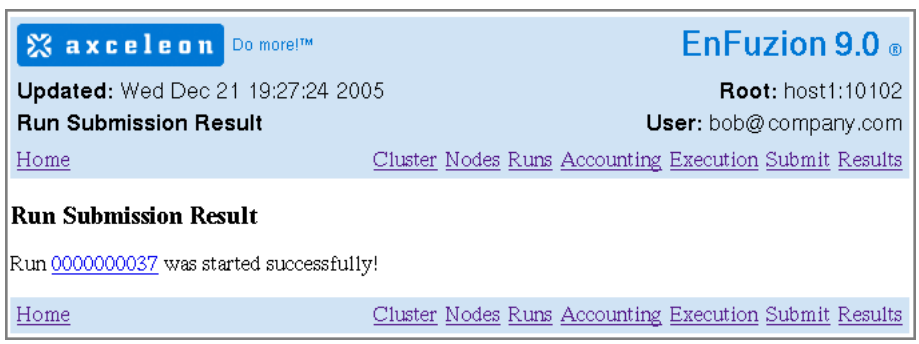
Clicking on the **Submit** button will upload the selected file and create a run from it. If your run file was not correctly formed, you will see an error message reporting that adding the run failed. Otherwise, a page will be displayed, enabling you to select and upload optional data files (see Figure 10-3).

Figure 10-3. Submission of Data Files



Select a file with the **Browse** button, and then click on the **Submit Data File** button. You will see the data file added in the list below the submission form. Repeat this process for every data file, and select **Start Run Execution**. The results of starting a run will then be displayed (see Figure 10-4).

Figure 10-4. Successful Run Submission



If the run was successfully started, you can immediately view its state by clicking on the link that includes the ID of the started run. This process is described with more detail in the Section called *Detailed Run Information Page*.

Note that although EnFuzion allows you to specify a custom name for the run directory, custom directories are not supported by the Eye. You need to allow the run to create its own directory, using a default name.

If you are accessing the Eye via a proxy, it is possible that the proxy will not allow you to post large data files to the Eye. One solution is to bypass the proxy and connect to the Eye directly. Otherwise, you may need to contact your proxy administrator for assistance.

Monitoring Execution

This collection of pages displays an in-depth view of the EnFuzion cluster that the Eye is connected to, including its runs and nodes.

Cluster Status Page

The first table contains general information about the cluster (see Figure 10-5):

Figure 10-5. The Cluster Status Page

axceleon
Do more!™:
EnFuzion 9.0 ®
Updated: Wed Dec 21 19:11:08 2005
Root: host1:10102
Cluster Summary
User: anonymous
Home
Cluster Nodes Runs Accounting Execution Submit Results
Cluster host1:10102
Cluster Status Uptime Active Nodes Down Nodes Submitted Runs Completed Runs
host1:10102 Running 00:00:30 2 0 1 0
Nodes
Active Executing Idle Busy Down
Nodes 2 2 0 0 0
Runs
Created Started Stopped Done Failed
Runs 0 1 0 0 0
Messages
Time Object Message
Wed Dec 21 19:10:38 2005 Cluster host1:10102 reading user accounts from file "C:\EnFuzion\config\user.accounts.txt"
Wed Dec 21 19:10:38 2005 Cluster host1:10102 reading nodes from file "C:\EnFuzion\config\enfuzion.nodes.txt"
Wed Dec 21 19:10:38 2005 Cluster host1:10102 a license for 1024 nodes has been found in file "C:\EnFuzion\config\enflicense.txt"
Wed Dec 21 19:10:38 2005 Cluster host1:10102 reading root options from file "C:\EnFuzion\config\root.options.txt"
Wed Dec 21 19:10:38 2005 Cluster host1:10102 build 9.0.002 for WindowsNT
Show complete log
Home
Cluster Nodes Runs Accounting Execution Submit Results

- Cluster: the host name and port that the EnFuzion Dispatcher is using
- Status: the status of the cluster
- Uptime: the total time that the cluster has been running
- Active Nodes: the number of active nodes, these might be executing or idle
- Down Nodes: the number of nodes that are down and unable to perform work

- Submitted Runs: the number of runs already submitted to the cluster
- Completed Runs: the number of runs completed by the cluster

The "Nodes" link takes you to the Node List page, as described in the Section called *Node List Page* below. The corresponding table shows the numbers of nodes, grouped by the node status.

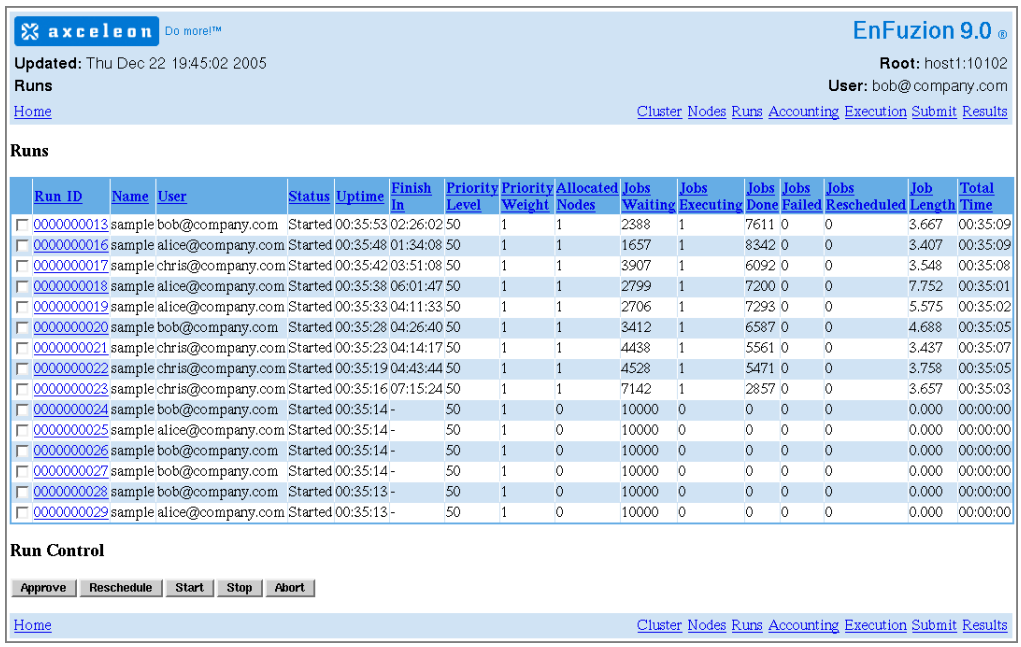
By following the "Runs" link, a list of runs is requested. See the Section called *Run List Page*. The corresponding table shows the number of runs, grouped by their status.

Finally, a table lists the ten most recent diagnostic messages from the cluster log that merit user attention. If there are more than ten messages, two buttons under the table take allow you to view all diagnostic messages or the complete cluster log, respectively.

Run List Page

This page displays a single table, containing all of the runs that the EnFuzion cluster recognizes. The following information is displayed in the table (see Figure 10-6):

Figure 10-6. The Run List Page


axceleon
Do more!™
EnFuzion 9.0
Updated: Thu Dec 22 19:45:02 2005
Root: host1:10102
Runs
User: bob@company.com
Home
Cluster Nodes Runs Accounting Execution Submit Results
Runs
Run ID Name User Status Uptime Finish In Priority Level Priority Weight Allocated Nodes Jobs Waiting Jobs Executing Jobs Done Jobs Failed Jobs Rescheduled Job Length Total Time
0000000013 sample bob@company.com Started 00:35:53 02:26:02 50 1 1 2388 1 7611 0 0 3.667 00:35:09
0000000016 sample alice@company.com Started 00:35:48 01:34:08 50 1 1 1657 1 8342 0 0 3.407 00:35:09
0000000017 sample chris@company.com Started 00:35:42 03:51:08 50 1 1 3907 1 6092 0 0 3.548 00:35:08
0000000018 sample alice@company.com Started 00:35:38 06:01:47 50 1 1 2799 1 7200 0 0 7.752 00:35:01
0000000019 sample alice@company.com Started 00:35:33 04:11:33 50 1 1 2706 1 7293 0 0 5.575 00:35:02
0000000020 sample bob@company.com Started 00:35:28 04:26:40 50 1 1 3412 1 6587 0 0 4.688 00:35:05
0000000021 sample chris@company.com Started 00:35:23 04:14:17 50 1 1 4438 1 5561 0 0 3.437 00:35:07
0000000022 sample chris@company.com Started 00:35:19 04:43:44 50 1 1 4528 1 5471 0 0 3.758 00:35:05
0000000023 sample chris@company.com Started 00:35:16 07:15:24 50 1 1 7142 1 2857 0 0 3.657 00:35:03
0000000024 sample bob@company.com Started 00:35:14 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
0000000025 sample alice@company.com Started 00:35:14 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
0000000026 sample bob@company.com Started 00:35:14 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
0000000027 sample bob@company.com Started 00:35:14 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
0000000028 sample bob@company.com Started 00:35:13 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
0000000029 sample alice@company.com Started 00:35:13 - 50 1 0 10000 0 0 0 0 0.000 00:00:00
Run Control
Approve Reschedule Start Stop Abort
Home
Cluster Nodes Runs Accounting Execution Submit Results

- Selection: the first column allows you to add and remove runs from the selection
- Run ID: the run ID. Clicking this takes you to the detailed run information page.
- Name: the run name

- User: user ID of the run owner
- Status: the run status
- Uptime: the time elapsed since the run was started
- Finish In: the estimated time required to complete this run
- Priority Level: priority level for the run
- Priority Weight: priority weight for the run
- Allocated Nodes: the number of nodes allocated to perform work for this run
- Jobs Waiting: the number of jobs still waiting to be executed
- Jobs Executing: the number of jobs currently executing
- Jobs Done: the number of jobs completed successfully
- Jobs Failed: the number of jobs that did not complete due to some error
- Jobs Rescheduled: the number of times that the jobs from the run were rescheduled
- Job Length: the average time to complete a job
- Total Time: the sum of completion times for all the jobs

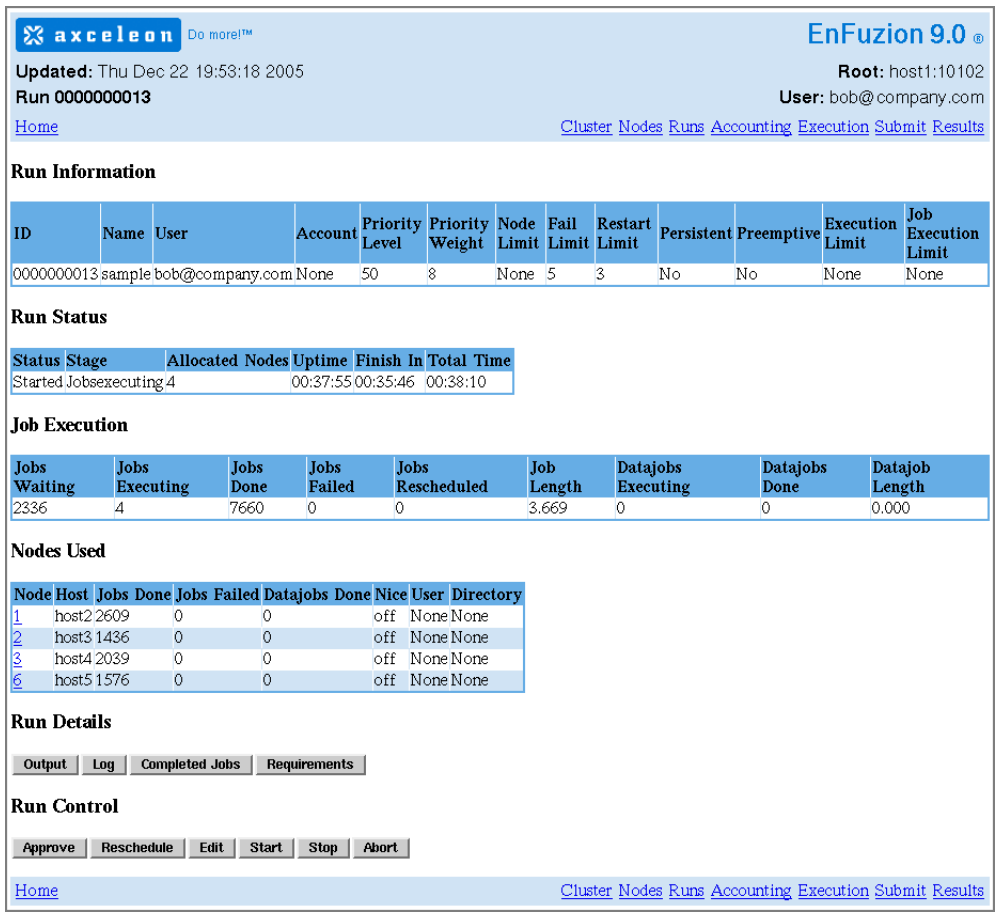
Below the table, the buttons under the Run Control section allow you to control the set of selected runs. Possible actions are:

- Approve: approve selected runs
- Reschedule: reschedule failed jobs from selected runs
- Start: start selected runs
- Stop: stop selected runs
- Abort: abort selected runs

Detailed Run Information Page

This page displays detailed information about a single run (see Figure 10-7):

Figure 10-7. Detailed Run Information



axceleon Do more!™ EnFuzion 9.0 ®

Updated: Thu Dec 22 19:53:18 2005 **Root:** host1:10102

Run 0000000013 **User:** bob@company.com

Home Cluster Nodes Runs Accounting Execution Submit Results

Run Information

ID	Name	User	Account	Priority Level	Priority Weight	Node Limit	Fail Limit	Restart Limit	Persistent	Preemptive	Execution Limit	Job Execution Limit
0000000013	sample	bob@company.com	None	50	8	None	5	3	No	No	None	None

Run Status

Status	Stage	Allocated Nodes	Uptime	Finish In	Total Time
Started	Jobsexecuting	4	00:37:55	00:35:46	00:38:10

Job Execution

Jobs Waiting	Jobs Executing	Jobs Done	Jobs Failed	Jobs Rescheduled	Job Length	Datajobs Executing	Datajobs Done	Datajob Length
2336	4	7660	0	0	3.669	0	0	0.000

Nodes Used

Node	Host	Jobs Done	Jobs Failed	Datajobs Done	Nice	User	Directory
1	host2	2609	0	0	off	None	None
2	host3	1436	0	0	off	None	None
3	host4	2039	0	0	off	None	None
6	host5	1576	0	0	off	None	None

Run Details

Output Log Completed Jobs Requirements

Run Control

Approve Reschedule Edit Start Stop Abort

Home Cluster Nodes Runs Accounting Execution Submit Results

The first table contains general run information:

- Run ID: the run ID
- Name: the run name
- User: user ID of the run owner
- Account: user specified string
- Priority Level: priority level for the run
- Priority Weight: priority weight for the run
- Node Limit: maximum number of nodes to execute the run
- Fail Limit: maximum number of allowed failed jobs on a node
- Restart Limit: maximum number of times a job can be rescheduled

- Persistent: persistence switch
- Preemptive: preemption switch
- Execution Limit: time to complete the run
- Job Execution Limit: time to complete a job

The second table contains information about run status:

- Status: one of Created, Started, Done, Failed, Stopped
- Stage: one of Initializing, Rootstart, Jobsexecuting, Nodefinish, Rootfinish
- Allocated Nodes: the number of nodes allocated to perform work for this run.
- Uptime: the time elapsed since the run was started.
- Finish In: the estimated time required to complete this run.
- Total Time: the sum of completion times for all the jobs

The next table contains information about how the run is executing:

- Jobs Waiting: the number of jobs still waiting to be executed
- Job Executing: the number of jobs currently executing
- Jobs Done: the number of completed jobs
- Jobs Failed: the number of jobs that did not complete due to some error
- Jobs Rescheduled: the number of times that a job from the run was rescheduled
- Job Length: the average time to complete a job
- Datajobs Executing: the number of data jobs currently executing
- Datajobs Done: the number of completed data jobs
- Datajob Length: the average time to complete a data job

Below this table, a list of nodes that are initialized to serve this run is displayed. The following columns are specified for each node:

- Node: node ID
- Host: host name executing the node
- Jobs Done: jobs completed on the node, including successful and failed jobs
- Jobs Failed: jobs failed on the node
- Datajobs Done: datajobs completed on the node
- Nice: job priority on the node. If nice is on, then the jobs are executed at a background priority.
- User: user account on the node that executes the jobs

- Directory: the main directory where jobs are executing

At the bottom of the page, additional buttons enable you to view further run details:

- Output: shows list of files produced by this run
- Log: displays the run log
- Completed Jobs: takes you to the Completed Jobs page
- Requirements: lets you inspect and edit run requirements

Run requirements are shown in a list on a dedicated page: you may select and remove them with the "Remove" button or you may type a new requirement in the text field below the list and add it with the "Add" button.

The last row of buttons allows you to control the run. Possible actions are:

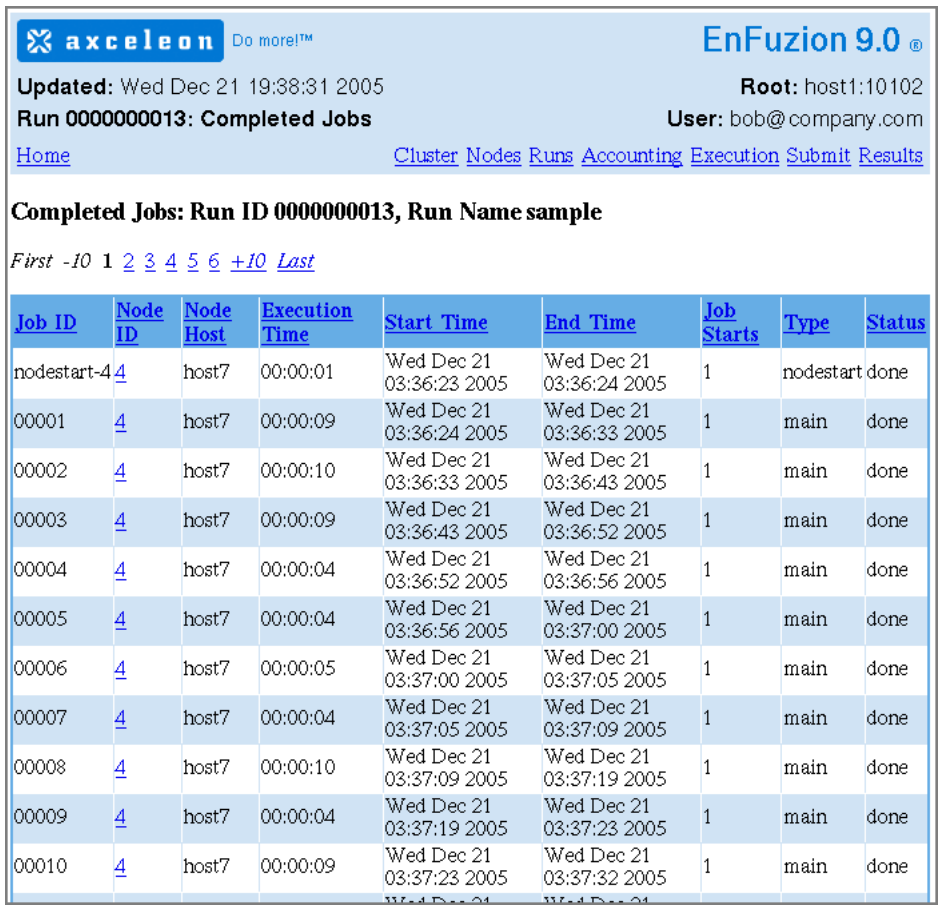
- Approve: approve the run
- Reschedule: reschedule failed jobs
- Edit: edit various run attributes
- Start: start the run
- Stop: stop the run
- Abort: abort the run

Editing run attributes brings you to a new page where you can edit the following run attributes: Priority Level, Priority Weight, Node Limit and Execution Limit. When you change these to the desired values, simply click on the "Apply Changes" button in order to commit the changes and have them take effect.

Completed Jobs Page

The completed jobs page shows a table of all jobs in the specified run that have completed (see Figure 10-8).

Figure 10-8. The Completed Jobs Page



axceleon Do more!™ EnFuzion 9.0 ®

Updated: Wed Dec 21 19:38:31 2005 **Root:** host1:10102

Run 0000000013: Completed Jobs **User:** bob@company.com

Home Cluster Nodes Runs Accounting Execution Submit Results

Completed Jobs: Run ID 0000000013, Run Name sample

First -10 **1** 2 3 4 5 6 *+10* *Last*

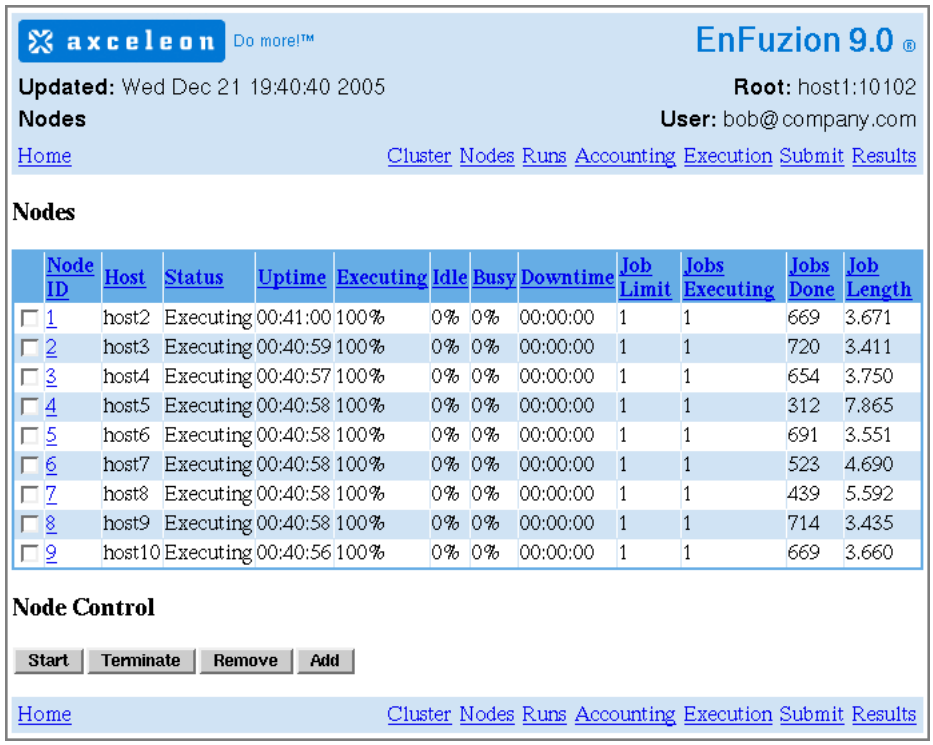
Job ID	Node ID	Node Host	Execution Time	Start Time	End Time	Job Starts	Type	Status
nodestart-4	4	host7	00:00:01	Wed Dec 21 03:36:23 2005	Wed Dec 21 03:36:24 2005	1	nodestart	done
00001	4	host7	00:00:09	Wed Dec 21 03:36:24 2005	Wed Dec 21 03:36:33 2005	1	main	done
00002	4	host7	00:00:10	Wed Dec 21 03:36:33 2005	Wed Dec 21 03:36:43 2005	1	main	done
00003	4	host7	00:00:09	Wed Dec 21 03:36:43 2005	Wed Dec 21 03:36:52 2005	1	main	done
00004	4	host7	00:00:04	Wed Dec 21 03:36:52 2005	Wed Dec 21 03:36:56 2005	1	main	done
00005	4	host7	00:00:04	Wed Dec 21 03:36:56 2005	Wed Dec 21 03:37:00 2005	1	main	done
00006	4	host7	00:00:05	Wed Dec 21 03:37:00 2005	Wed Dec 21 03:37:05 2005	1	main	done
00007	4	host7	00:00:04	Wed Dec 21 03:37:05 2005	Wed Dec 21 03:37:09 2005	1	main	done
00008	4	host7	00:00:10	Wed Dec 21 03:37:09 2005	Wed Dec 21 03:37:19 2005	1	main	done
00009	4	host7	00:00:04	Wed Dec 21 03:37:19 2005	Wed Dec 21 03:37:23 2005	1	main	done
00010	4	host7	00:00:09	Wed Dec 21 03:37:23 2005	Wed Dec 21 03:37:32 2005	1	main	done

- Job ID: ID of the job
- Node ID: ID of the node that the job was completed on
- Node Host: host name of the node that the job was completed on
- Execution Time: time that the job executed
- Start Time: time when the job was first started
- End Time: time when the job was completed
- Job Starts: number of times the job was started
- Type: type of job, which is either "nodestart" for jobs that initialize a node and "main" for user specified jobs
- Status: status of job, which is either "done" or "failed"

Node List Page

This page displays a list of all nodes that the EnFuzion cluster recognizes. For each node, the following information is displayed (see Figure 10-9):

Figure 10-9. The Node List Page


axceleon
Do more!™
EnFuzion 9.0 ®
Updated: Wed Dec 21 19:40:40 2005
Root: host1:10102
Nodes
User: bob@company.com
Home
Cluster Nodes Runs Accounting Execution Submit Results
Nodes
Node ID | Host | Status | Uptime | Executing | Idle | Busy | Downtime | Job Limit | Jobs Executing | Jobs Done | Job Length
1 host2 Executing 00:41:00 100% 0% 0% 00:00:00 1 1 669 3.671
2 host3 Executing 00:40:59 100% 0% 0% 00:00:00 1 1 720 3.411
3 host4 Executing 00:40:57 100% 0% 0% 00:00:00 1 1 654 3.750
4 host5 Executing 00:40:58 100% 0% 0% 00:00:00 1 1 312 7.865
5 host6 Executing 00:40:58 100% 0% 0% 00:00:00 1 1 691 3.551
6 host7 Executing 00:40:58 100% 0% 0% 00:00:00 1 1 523 4.690
7 host8 Executing 00:40:58 100% 0% 0% 00:00:00 1 1 439 5.592
8 host9 Executing 00:40:58 100% 0% 0% 00:00:00 1 1 714 3.435
9 host10 Executing 00:40:56 100% 0% 0% 00:00:00 1 1 669 3.660
Node Control
Start
Terminate
Remove
Add
Home
Cluster Nodes Runs Accounting Execution Submit Results

- Selection: the first column allows you to add and remove nodes from the selection
- Node ID: the node name
- Host: the host name of the node
- Status: one of Executing, Idle, Busy, Down, Starting, Terminating
- Uptime: the time elapsed since the node last changed its status to "Up"
- Executing: the percentage of the uptime that the node was executing user jobs
- Idle: the percentage of the uptime that the node was idle
- Busy: the percentage of the uptime that the node was unavailable, since it was busy with processing unrelated to EnFuzion

- Downtime: the time elapsed since the node last changed its status to "Down"
- Job Limit: the maximum number of concurrent jobs that this node can execute
- Jobs Executing: the number of jobs currently executing on this node
- Jobs Done: the number of jobs completed by this node
- Job Length: the average time needed to complete a job on this node

Clicking on the node name link provides you with yet more information about that node. See the Section called *Detailed Node Information page* for further information.

Below the table, you may choose to start, terminate or remove selected nodes or add a new node.

Adding a node brings you to a new page where you have to enter information on the new node. This data is mostly the same as the one used in the **enfuzion.nodes** file:

- Host name of the node
- Username used to login to the node
- Password that is used to login to the node. You need to type it twice in order to confirm it. If you use the key authorization for the SSH method, which does not require a password, just use the **dummy** string for the password.
- Connection type

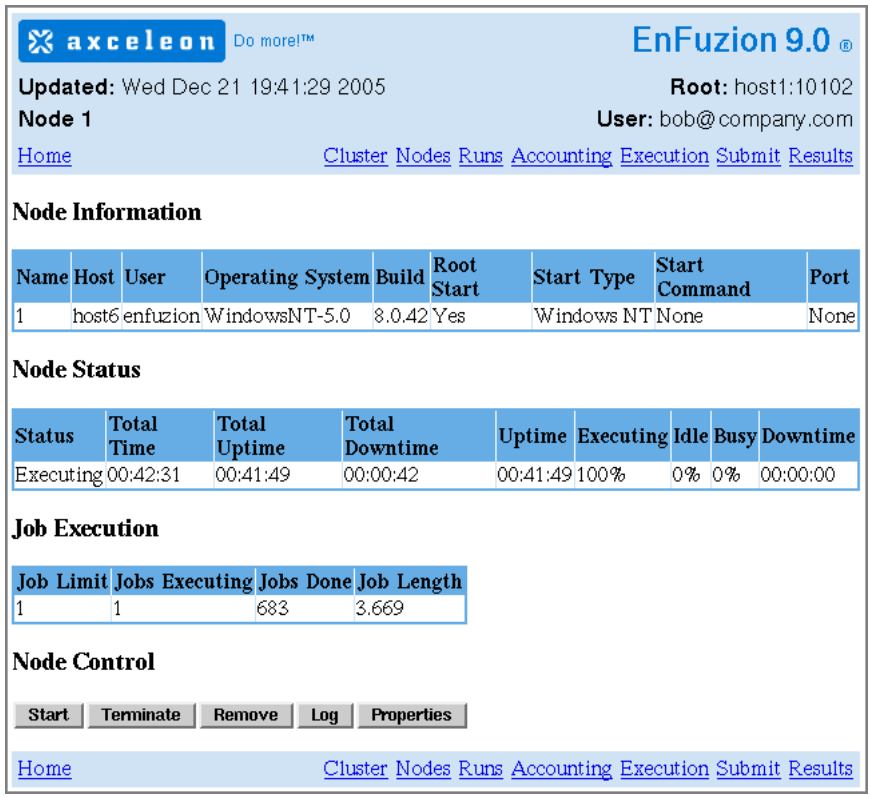
Clicking the "Add" button will add a new node to the cluster. You are only allowed to add a node, if privileges are not enforced or if you are logged in as a user with administrator privileges.

Detailed Node Information page

The detailed information page consists of three tables. The first table displays general information about

the selected node (see Figure 10-10):

Figure 10-10. Detailed Node Information


axceleon
Do more!™
EnFuzion 9.0 ®
Updated: Wed Dec 21 19:41:29 2005
Root: host1:10102
Node 1
User: bob@company.com
Home
Cluster Nodes Runs Accounting Execution Submit Results
Node Information
Name Host User Operating System Build Root Start Start Type Start Command Port
1 host6 enfuzion WindowsNT-5.0 8.0.42 Yes Windows NT None None
Node Status
Status Total Time Total Uptime Total Downtime Uptime Executing Idle Busy Downtime
Executing 00:42:31 00:41:49 00:00:42 00:41:49 100% 0% 0% 00:00:00
Job Execution
Job Limit Jobs Executing Jobs Done Job Length
1 1 683 3.669
Node Control
Start Terminate Remove Log Properties
Home
Cluster Nodes Runs Accounting Execution Submit Results

- Node ID: the node name
- Host: the host name of the node
- User: the user that is used to log on the node
- Port: the port used for communication with the EnFuzion Dispatcher
- Operating System: the operating system running on this node
- Root Start: switch to indicate whether the root starts the node or is the node started independently
- Start Type: the method to start the node
- Start Command: the command used to start the node

The second table displays the node's status information:

- Status: the node status

- Total Time: the total time since the node was added to the cluster
- Total Uptime: the total time that the node was "Up"
- Total Downtime: the total time that the node was "Down"
- Uptime: the time elapsed since the node last changed its status to "Up"
- Executing: the percentage of the uptime that the node was executing user jobs
- Idle: the percentage of the uptime that the node was idle
- Busy: the percentage of the uptime that the node was unavailable, since it was busy with processing unrelated to EnFuzion
- Downtime: the time elapsed since the node last changed its status to "Down"

Finally, the third table displays job execution statistics for this node:

- Job Limit: the maximum number of concurrent jobs that this node can execute
- Jobs Executing: the number of jobs currently executing on this node
- Jobs Done: the number of jobs completed by this node
- Job Length: the average time needed to complete a job on this node

Below the tables, a set of buttons enables you to control the node. You may choose to:

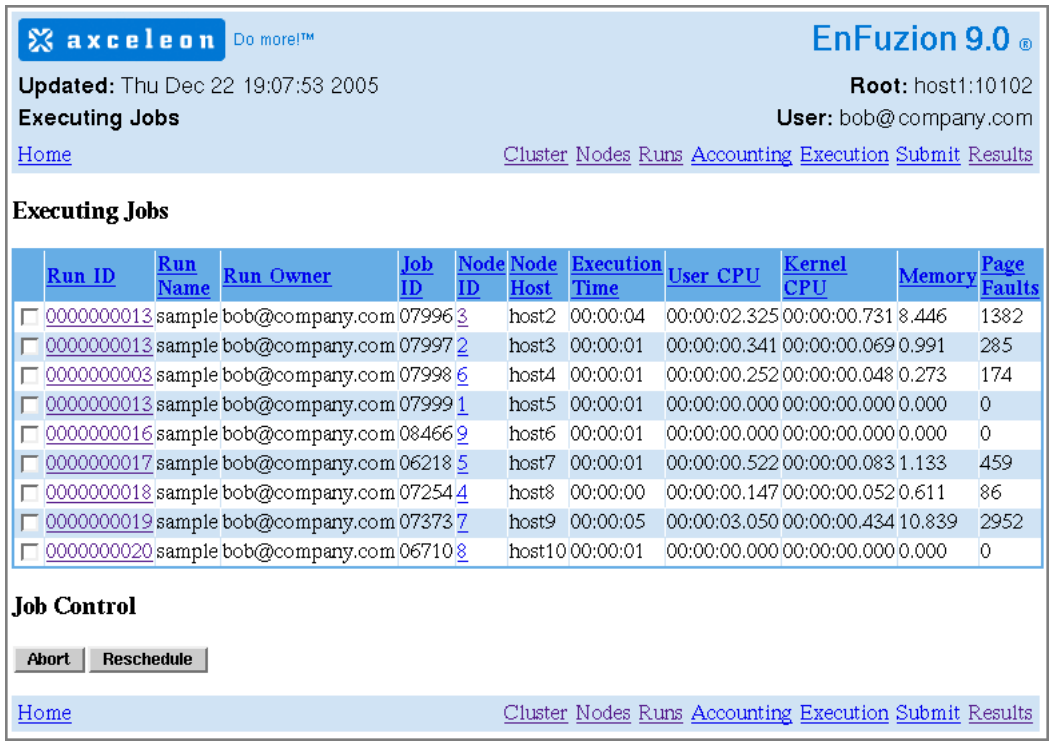
- Start the node
- Terminate the node
- Remove the node
- View the log
- Edit the node properties

Selecting the properties button brings you to the Node Properties page. Here you can view all the node properties in a list. You may select any number of properties and remove them using the "Remove" button or enter a new property in the text field below the "Remove" button and add it with the "Add" button.

Executing Jobs Page

The Executing Jobs page shows all currently executing jobs. The table consists of the following fields:

Figure 10-11. The Executing Jobs Page



Run ID	Run Name	Run Owner	Job ID	Node ID	Node Host	Execution Time	User CPU	Kernel CPU	Memory	Page Faults
0000000013	sample	bob@company.com	07996	3	host2	00:00:04	00:00:02.325	00:00:00.731	8.446	1382
0000000013	sample	bob@company.com	07997	2	host3	00:00:01	00:00:00.341	00:00:00.069	0.991	285
0000000003	sample	bob@company.com	07998	6	host4	00:00:01	00:00:00.252	00:00:00.048	0.273	174
0000000013	sample	bob@company.com	07999	1	host5	00:00:01	00:00:00.000	00:00:00.000	0.000	0
0000000016	sample	bob@company.com	08466	9	host6	00:00:01	00:00:00.000	00:00:00.000	0.000	0
0000000017	sample	bob@company.com	06218	5	host7	00:00:01	00:00:00.522	00:00:00.083	1.133	459
0000000018	sample	bob@company.com	07254	4	host8	00:00:00	00:00:00.147	00:00:00.052	0.611	86
0000000019	sample	bob@company.com	07373	7	host9	00:00:05	00:00:03.050	00:00:00.434	10.839	2952
0000000020	sample	bob@company.com	06710	8	host10	00:00:01	00:00:00.000	00:00:00.000	0.000	0

- Selection: allows you to add or remove jobs to the selected set
- Run ID: shows ID of the run the job belongs to. The ID links to the respective run page
- Run Name: shows name of the run the job belongs to
- Run Owner: shows the user ID of the run owner
- Job ID: shows ID of the job
- Node ID: ID of the node the job is currently executing on; the ID links to the respective node page
- Node Host: hostname of that node
- Execution Time: the wall clock time the job has been executing for
- User CPU: the CPU usage in the user space
- Kernel CPU: the CPU usage in the kernel space
- Memory: maximum memory usage in Kb

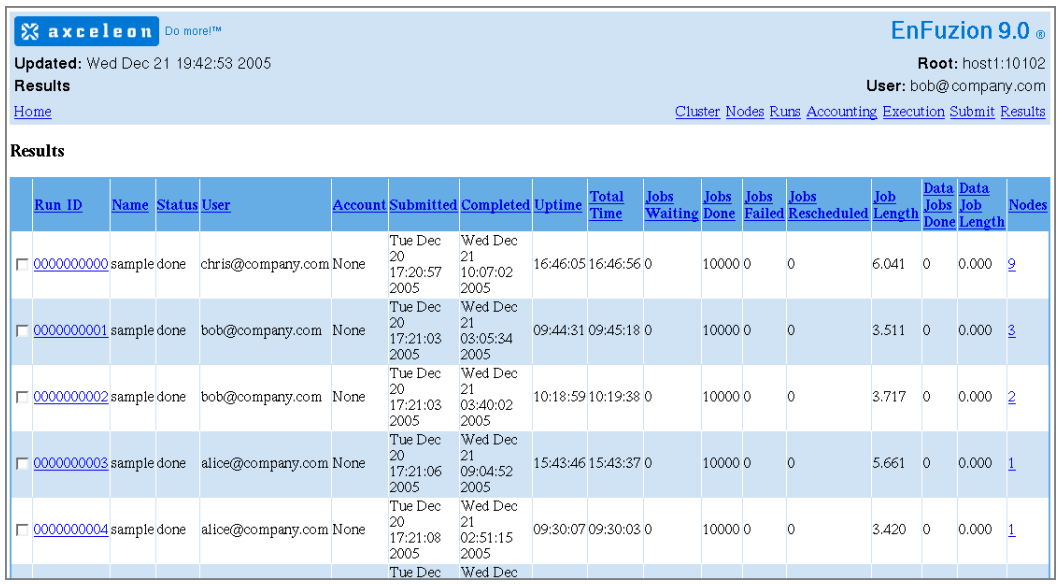
- Page Faults: number of page faults

Below the table two buttons allow you to abort or reschedule the selected set of jobs.

Run Results page

This page shows a list of all directories that store results of an EnFuzion run (see Figure 10-12):

Figure 10-12. The Run Results page



	Run ID	Name	Status	User	Account	Submitted	Completed	Uptime	Total Time	Jobs Waiting	Jobs Done	Jobs Failed	Jobs Rescheduled	Job Length	Data Jobs Done	Data Job Length	Nodes
☐	0000000000	sample	done	chris@company.com	None	Tue Dec 20 17:20:57 2005	Wed Dec 21 10:07:02 2005	16:46:05	16:46:56	0	10000	0	0	6.041	0	0.000	9
☐	0000000001	sample	done	bob@company.com	None	Tue Dec 20 17:21:03 2005	Wed Dec 21 03:05:34 2005	09:44:31	09:45:18	0	10000	0	0	3.511	0	0.000	3
☐	0000000002	sample	done	bob@company.com	None	Tue Dec 20 17:21:03 2005	Wed Dec 21 03:40:02 2005	10:18:59	10:19:38	0	10000	0	0	3.717	0	0.000	2
☐	0000000003	sample	done	alice@company.com	None	Tue Dec 20 17:21:06 2005	Wed Dec 21 09:04:52 2005	15:43:46	15:43:37	0	10000	0	0	5.661	0	0.000	1
☐	0000000004	sample	done	alice@company.com	None	Tue Dec 20 17:21:08 2005	Wed Dec 21 02:51:15 2005	09:30:07	09:30:03	0	10000	0	0	3.420	0	0.000	1
						Tue Dec	Wed Dec										

- Selection: the first column allows you to add and remove completed runs from the selection
- Run ID: ID of the completed run, the ID links to the page with the contents of the run directory where output files are stored
- Name: name of the run
- Status: "done" or "failed"
- User: the owner of the run
- Account: user specified string
- Submitted: time of submission of run
- Completed: time of completion of run
- Uptime: time the run was up

- Total Time: the sum of execution times for all the nodes
- Jobs Waiting: number of waiting jobs. If the run is aborted, then this number represents the number of uncompleted jobs
- Jobs Done: number of done jobs
- Jobs Failed: number of failed jobs
- Jobs Rescheduled: number of rescheduled jobs
- Job Length: average length of a single job
- Data Jobs Done: number of done data jobs
- Data Job Length: average length of a single data job
- Nodes: number of used nodes, which links to the page of nodes used.

Beneath the table, the buttons in the Run Details section allow you to inspect details of the selected run:

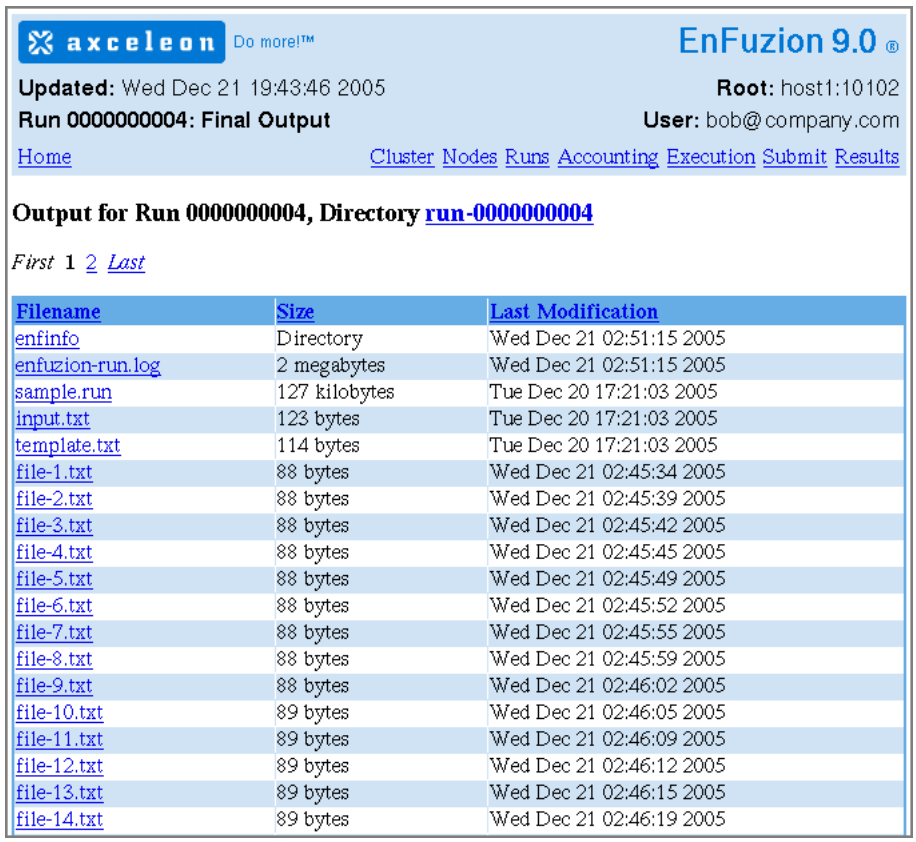
- Output: shows contents of the directory containing files output by the run
- Log: shows run log
- Completed Jobs: takes you to the Completed Jobs page
- Used Nodes: shows the Used Nodes page

The buttons in the Run Control section allow you to control the run:

- Reschedule: restarts the run. Failed jobs are submitted for execution, while successful jobs are not affected.
- Delete: deletes the run directory and all user files in the directory. This operation deletes all information about the run, use with care and only in extreme cases!

By following the Run ID link or using the output button, the user may browse the contents of run directories in a similar fashion to browsing a file system with a file manager. Clicking on a directory displays its contents, their sizes and the dates of their last modification (see Figure 10-13):

Figure 10-13. Run Directory

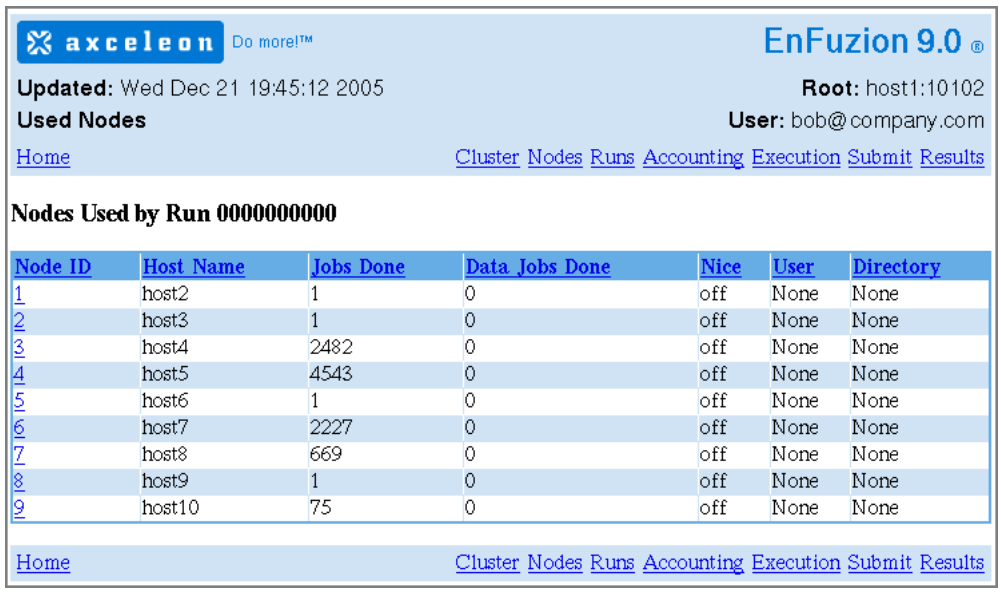

axceleon
Do more!™
EnFuzion 9.0 ®
Updated: Wed Dec 21 19:43:46 2005
Root: host1:10102
Run 0000000004: Final Output
User: bob@company.com
Home
Cluster Nodes Runs Accounting Execution Submit Results
Output for Run 0000000004, Directory run-0000000004
First 1 2 Last
Filename | Size | Last Modification
enfinfo | Directory | Wed Dec 21 02:51:15 2005
enfuzion-run.log | 2 megabytes | Wed Dec 21 02:51:15 2005
sample.run | 127 kilobytes | Tue Dec 20 17:21:03 2005
input.txt | 123 bytes | Tue Dec 20 17:21:03 2005
template.txt | 114 bytes | Tue Dec 20 17:21:03 2005
file-1.txt | 88 bytes | Wed Dec 21 02:45:34 2005
file-2.txt | 88 bytes | Wed Dec 21 02:45:39 2005
file-3.txt | 88 bytes | Wed Dec 21 02:45:42 2005
file-4.txt | 88 bytes | Wed Dec 21 02:45:45 2005
file-5.txt | 88 bytes | Wed Dec 21 02:45:49 2005
file-6.txt | 88 bytes | Wed Dec 21 02:45:52 2005
file-7.txt | 88 bytes | Wed Dec 21 02:45:55 2005
file-8.txt | 88 bytes | Wed Dec 21 02:45:59 2005
file-9.txt | 88 bytes | Wed Dec 21 02:46:02 2005
file-10.txt | 89 bytes | Wed Dec 21 02:46:05 2005
file-11.txt | 89 bytes | Wed Dec 21 02:46:09 2005
file-12.txt | 89 bytes | Wed Dec 21 02:46:12 2005
file-13.txt | 89 bytes | Wed Dec 21 02:46:15 2005
file-14.txt | 89 bytes | Wed Dec 21 02:46:19 2005

You can browse recursively through the subdirectories and download the files contained in them.

Used Nodes Page

This page shows a table of all nodes used by the specified run (see Figure 10-14):

Figure 10-14. The Used Nodes Page

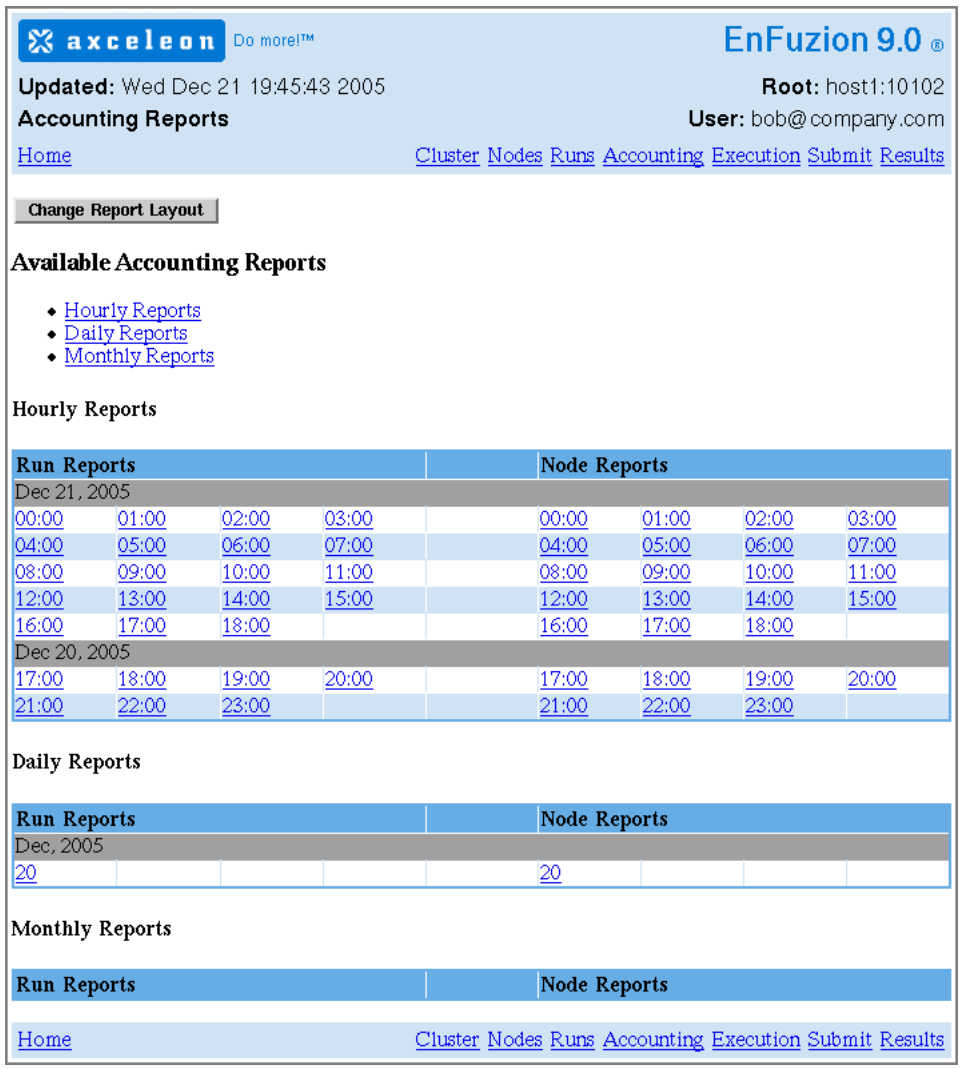

axceleon
Do more!™
EnFuzion 9.0 ®
Updated: Wed Dec 21 19:45:12 2005
Root: host1:10102
Used Nodes
User: bob@company.com
Home
Cluster Nodes Runs Accounting Execution Submit Results
Nodes Used by Run 0000000000
Node ID | Host Name | Jobs Done | Data Jobs Done | Nice | User | Directory
1 | host2 | 1 | 0 | off | None | None
2 | host3 | 1 | 0 | off | None | None
3 | host4 | 2482 | 0 | off | None | None
4 | host5 | 4543 | 0 | off | None | None
5 | host6 | 1 | 0 | off | None | None
6 | host7 | 2227 | 0 | off | None | None
7 | host8 | 669 | 0 | off | None | None
8 | host9 | 1 | 0 | off | None | None
9 | host10 | 75 | 0 | off | None | None
Home
Cluster Nodes Runs Accounting Execution Submit Results

- Node ID: node ID; it links to the appropriate node page
- Host Name: host name of the node
- Jobs Done: number of jobs completed on this node
- Data Jobs Done: number of data jobs completed on this node
- Nice: execution priority
- User: the account used on the node
- Directory: the working directory on the node

Accounting Page

The accounting page lists available run and node activity reports. At (see Figure 10-15):

Figure 10-15. The Accounting Page



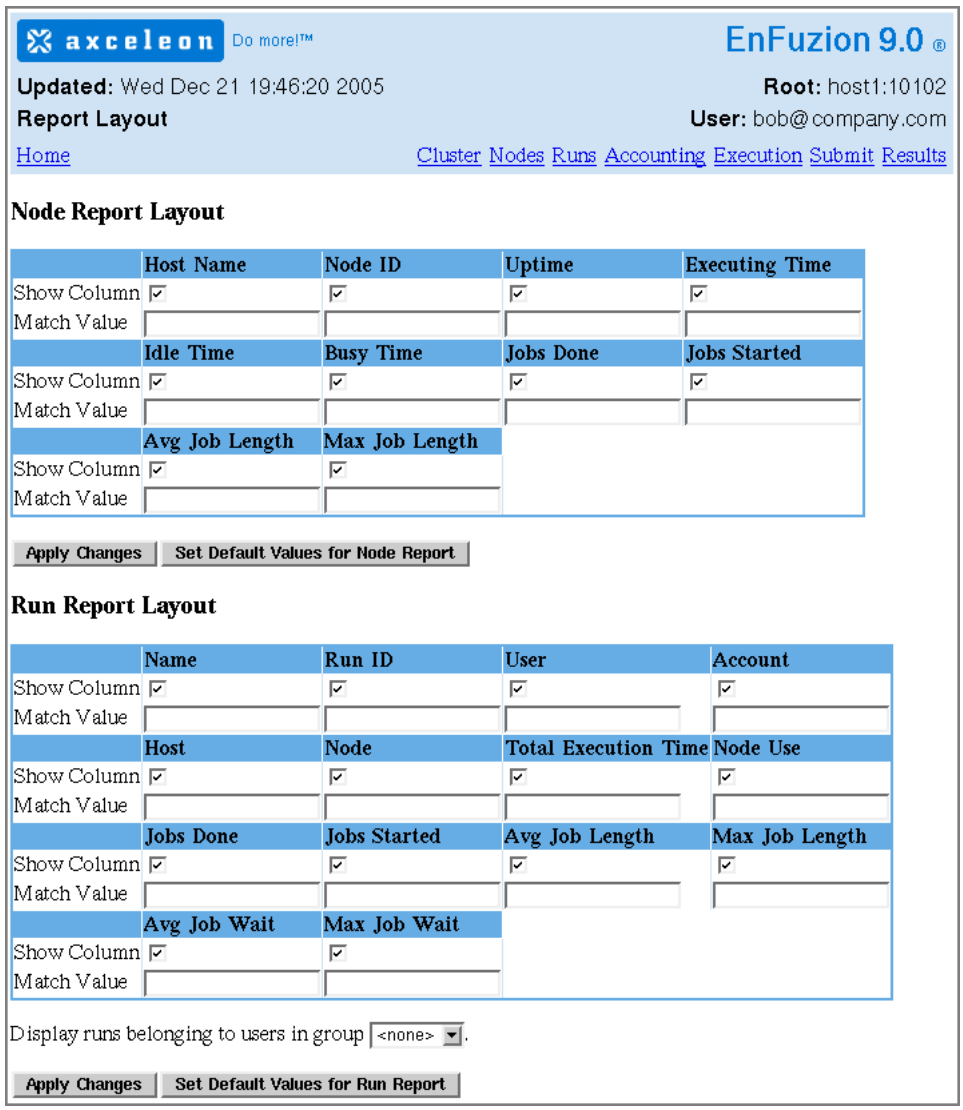
At the top of the page, a "Change Report Layout" button takes you to the Report Layout page and below it three links, "Hourly Reports", "Daily Reports" and "Monthly Reports" take you to the parts of the page listing hourly, daily and monthly reports, respectively.

The three tables below list reports by period of activity: run reports in the left column and node reports in the right column. First table lists hourly reports, the second one daily reports and the last table lists monthly reports: clicking on the links in the table shows the desired report.

Report Layout Page

The report layout page lets you to edit the columns shown in the run and node reports (see the Section called *Accounting Page*):

Figure 10-16. The Report Layout Page



The first table is dedicated to the node reports and the second one to the run reports. You may check the "Group By Column" check box in order to group report rows by certain columns. In this case, the values for grouped rows are added together. Entering a "Match Value" only shows the rows where the desired column's value matches the entered one.

You may use the buttons beneath each table to reset the layout specification to the default one.

At the bottom of the page, you may select a group filter for run reports. Only runs owned by users in the selected group will be shown in the reports.

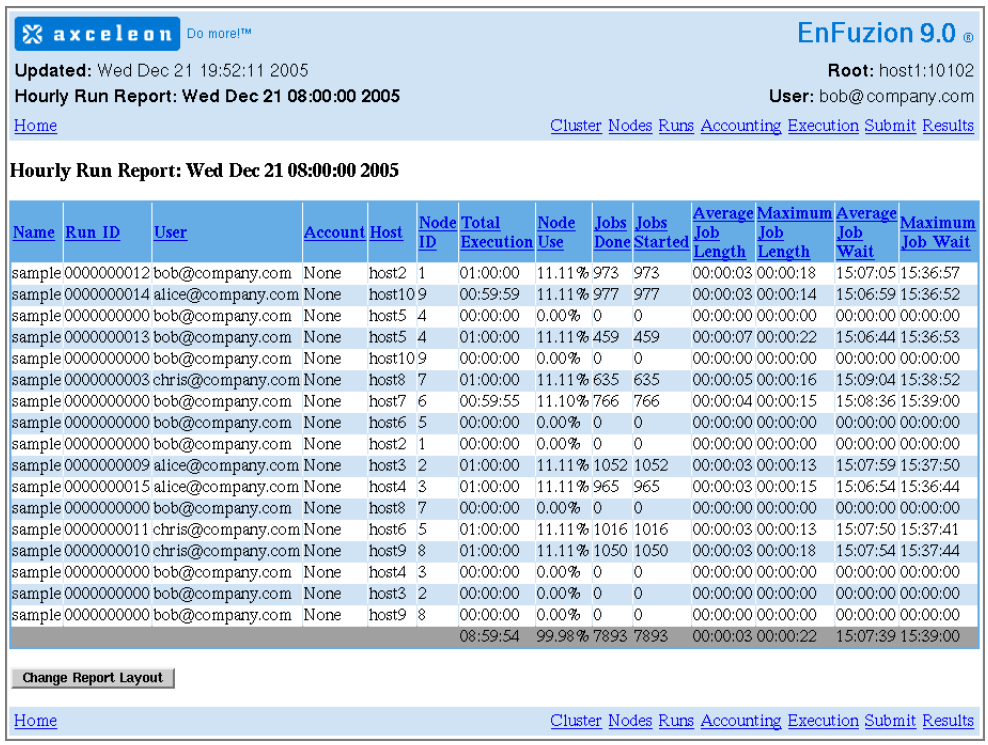
Changes to layout should be committed by clicking on the "Apply Changes" button.

Report Pages

Each report page starts with a header that describes the report type and the period for which the report stands. The actual report table follows and the page ends with a button that allows you to change the report layout.

Reports are available for runs (see Figure 10-17) and nodes (see Figure 10-18):

Figure 10-17. Run Report



axceleon Do more!™ EnFuzion 9.0 ®

Updated: Wed Dec 21 19:52:11 2005 **Root:** host1:10102

Hourly Run Report: Wed Dec 21 08:00:00 2005 **User:** bob@company.com

Home Cluster Nodes Runs Accounting Execution Submit Results

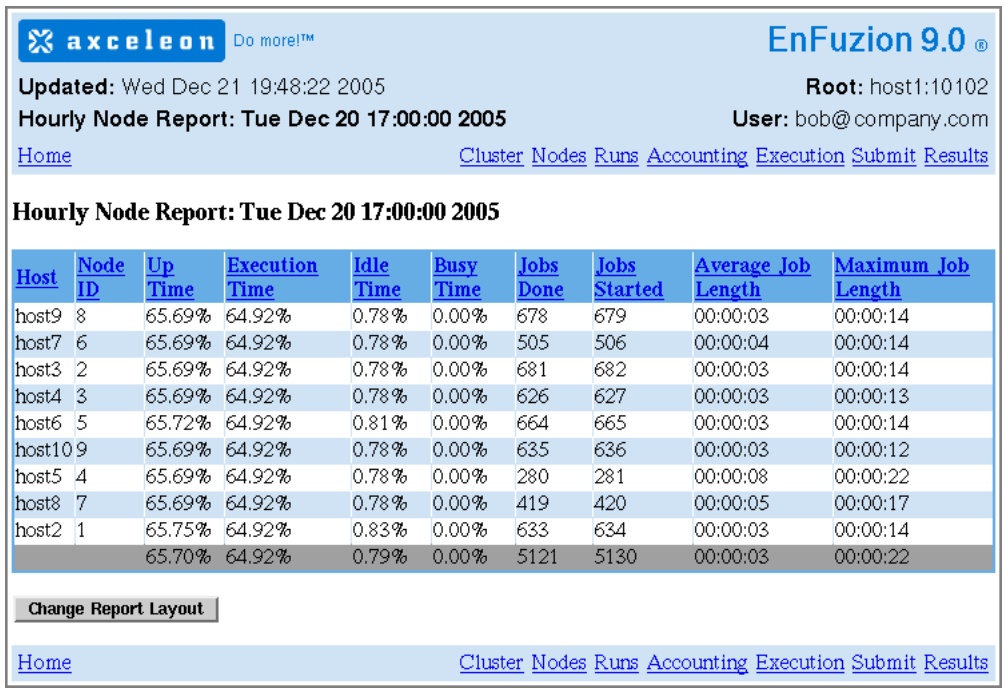
Hourly Run Report: Wed Dec 21 08:00:00 2005

Name	Run ID	User	Account	Host	Node ID	Total Execution	Node Use	Jobs Done	Jobs Started	Average Job Length	Maximum Job Length	Average Job Wait	Maximum Job Wait
sample	0000000012	bob@company.com	None	host2	1	01:00:00	11.11%	973	973	00:00:03	00:00:18	15:07:05	15:36:57
sample	0000000014	alice@company.com	None	host10	9	00:59:59	11.11%	977	977	00:00:03	00:00:14	15:06:59	15:36:52
sample	0000000000	bob@company.com	None	host5	4	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000013	bob@company.com	None	host5	4	01:00:00	11.11%	459	459	00:00:07	00:00:22	15:06:44	15:36:53
sample	0000000000	bob@company.com	None	host10	9	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000003	chris@company.com	None	host8	7	01:00:00	11.11%	635	635	00:00:05	00:00:16	15:09:04	15:38:52
sample	0000000000	bob@company.com	None	host7	6	00:59:55	11.10%	766	766	00:00:04	00:00:15	15:08:36	15:39:00
sample	0000000000	bob@company.com	None	host6	5	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000000	bob@company.com	None	host2	1	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000009	alice@company.com	None	host3	2	01:00:00	11.11%	1052	1052	00:00:03	00:00:13	15:07:59	15:37:50
sample	0000000015	alice@company.com	None	host4	3	01:00:00	11.11%	965	965	00:00:03	00:00:15	15:06:54	15:36:44
sample	0000000000	bob@company.com	None	host8	7	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000011	chris@company.com	None	host6	5	01:00:00	11.11%	1016	1016	00:00:03	00:00:13	15:07:50	15:37:41
sample	0000000010	chris@company.com	None	host9	8	01:00:00	11.11%	1050	1050	00:00:03	00:00:18	15:07:54	15:37:44
sample	0000000000	bob@company.com	None	host4	3	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000000	bob@company.com	None	host3	2	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
sample	0000000000	bob@company.com	None	host9	8	00:00:00	0.00%	0	0	00:00:00	00:00:00	00:00:00	00:00:00
						08:59:54	99.98%	7893	7893	00:00:03	00:00:22	15:07:39	15:39:00

Change Report Layout

Home Cluster Nodes Runs Accounting Execution Submit Results

Figure 10-18. Node Report



axceleon Do more!™ EnFuzion 9.0 ®

Updated: Wed Dec 21 19:48:22 2005 **Root:** host1:10102

Hourly Node Report: Tue Dec 20 17:00:00 2005 **User:** bob@company.com

Home Cluster Nodes Runs Accounting Execution Submit Results

Hourly Node Report: Tue Dec 20 17:00:00 2005

Host	Node ID	Up Time	Execution Time	Idle Time	Busy Time	Jobs Done	Jobs Started	Average Job Length	Maximum Job Length
host9	8	65.69%	64.92%	0.78%	0.00%	678	679	00:00:03	00:00:14
host7	6	65.69%	64.92%	0.78%	0.00%	505	506	00:00:04	00:00:14
host3	2	65.69%	64.92%	0.78%	0.00%	681	682	00:00:03	00:00:14
host4	3	65.69%	64.92%	0.78%	0.00%	626	627	00:00:03	00:00:13
host6	5	65.72%	64.92%	0.81%	0.00%	664	665	00:00:03	00:00:14
host10	9	65.69%	64.92%	0.78%	0.00%	635	636	00:00:03	00:00:12
host5	4	65.69%	64.92%	0.78%	0.00%	280	281	00:00:08	00:00:22
host8	7	65.69%	64.92%	0.78%	0.00%	419	420	00:00:05	00:00:17
host2	1	65.75%	64.92%	0.83%	0.00%	633	634	00:00:03	00:00:14
		65.70%	64.92%	0.79%	0.00%	5121	5130	00:00:03	00:00:22

Change Report Layout

Home Cluster Nodes Runs Accounting Execution Submit Results

Error Messages List

This section lists error messages that the Eye produces.

General Error

An unpredicted error occurred. Please follow the instructions on the page in order to try and remedy the problem. Retry your action and if it fails again, restart the Eye and retry your action again. If the problem persists, send a bug report with a detailed description of how to reproduce it to support@axceleon.com.

Error: Access Denied

The client has been denied access to the Eye. You should check you access permissions in the **root.options** file.

Error: Authentication Failed

The client failed to log in to the Eye and to the Dispatcher. Check that you have used a proper user identity file, generated by the enfcmd utility, and that the file has not been altered by anyone.

Error: Connection Failed

The Eye was unable to connect to the Dispatcher. Please verify that the Dispatcher is actually running and that the Eye has been setup to try and connect to the proper port.

Error: Empty Selection

You have attempted to perform an action that requires at least one selected item, but you have selected none.

Error: Multiple Selected Items Not Allowed

You have attempted to perform an action that requires exactly one selected item, but you have selected more than one.

Error: Action Not Permitted!

You have chosen an action that requires user privileges that you do not have: perhaps you have attempted to perform an administrative action while not logged in as a user with administrative privileges or have chosen to manipulate a run that is not owned by the user you are currently logged in as.

Error: The Eye has Quit

The Dispatcher was run in the batch mode and has exited, bringing down the Eye with him. You need to start the Eye manually if you wish to browse the run results after the Dispatcher has quit or set the **eyeterminate** option to off in **root.options** file which will prevent the Dispatcher from taking down the Eye when it exits.

Error: Login Failed

Your session has probably expired, please go to the Home Page and attempt to log in again.

Error: Dispatcher Not Found

The Eye was unable to connect to the Dispatcher. Check the port number of the Dispatcher given to the Eye through command line options or entered through the login page.

Error: No File Name

You have attempted to submit a file without specifying which file.

Error: No Such Node

You have attempted to display information about a node that does not exist.

Error: No Such Run

You have attempted to fetch information on a run that the Dispatcher does not recognize.

Error: No Run Results

The results of the requested run do not exist, or are in a directory with a non-default name.

Error: No Reporting Data

Reporting data for the period you want the report for was not found.

Error: Page Not Found

You have requested a page that the Eye knows nothing about.

Error: Session Limit Reached

The maximum number of concurrent sessions that the Eye is willing to handle has been reached. You need to wait for one session to expire. Currently, the Eye supports 256 sessions. A session expires after a week of inactivity.

Error: Run Submission Expired

The run submission has expired since you have not completed it in a reasonable time. The submission cannot be completed and you need to resubmit the run.

Error: Run Submission Failed

The Eye was unable to submit the run to the Dispatcher.

Error: Mandatory Parameters Missing

A parameter that is mandatory was not entered. This error might happen whenever the Eye requires you to supply some values for an action like editing node or run attributes, adding a node or similar.

Error: Invalid Parameter Value

A parameter you entered has a value that is not allowed.

Error: Passwords Do Not Match

When adding a node you need to enter the same password twice in order to confirm it. You have not entered the same password in both text fields.

Handling of Privileges

Root options **noanonsubmit**, see details in the Section called *Rejecting Anonymous Run Submission* in Chapter 6, and **privileges**, see details in the Section called *Enforcing Privileges* in Chapter 6, affect which actions can be performed by users. By default, **noanonsubmit** and **privileges** are turned off, which allows any action to be performed by any user.

If **noanonsubmit** is turned on, then the following action is not permitted by users with the anonymous user ID:

- run submission, described in the Section called *Submitting a Run*;

If **privileges** are turned on, then the following actions are permitted only by users with administrative privileges:

- Start, Terminate, Remove actions on the Node List page, described in the Section called *Node List Page*, and the Add Node action on the subpage to add a new node;
- Start, Terminate, Remove actions on the Detailed Node Information page, described in the Section called *Detailed Node Information page*; Remove, Add actions on the Properties subpage;

If **privileges** are turned on, then the following actions are permitted only by users with administrative privileges or the run owner:

- Approve, Reschedule, Start, Stop, Abort actions on the Run List page, and access to the link under the Run ID field. These are described in the Section called *Run List Page*;
- access to the Detailed Run Information page, described in the Section called *Detailed Run Information Page*, and its actions on the page;
- Abort, Reschedule actions on the Executing Jobs page, described in the Section called *Executing Jobs Page*;
- Output, Reschedule and Delete actions on the Results page, described in the Section called *Run Results page*, and access to the link under the Run ID field.

Access Control

The Eye offers IP-based authentication. The administrator can set a list of IP addresses that are allowed or denied to connect to the Eye (see the Section called *Restricting Access to the Eye* in Chapter 6 for details).

Command Line Interface

EnFuzion provides command line programs **enfsub** and **enfcmd** to communicate with the Dispatcher. The **enfsub** is primarily used to submit runs for execution. It is also able to handle user identity. The **enfcmd** program provides a complete set of commands to interact with the Dispatcher. Most common tasks are simplified with high level commands. A complete Dispatcher API is provided for other tasks. All **enfcmd** commands can be easily used in scripts.

The following sections describe **enfsub** and **enfcmd** in detail.

The Enfsub Program

The **enfsub** program has the following options:

```
enfsub  [ <options> ]  <program>  [ <program_options> ]
enfsub  [ <options> ]  <script>  [ <script_options> ]
enfsub  [ <options> ]  [ -run ]  <run_file>  [ <input_files> ]
```

The program is used to submit the run for execution as a command line program, a script or a parametric execution, respectively.

<options> are:

- **-attach <run_ID>**

  attach to an existing run with the **<run_ID>** ID.

- **[-account | -a] <name>**

  a user specified string that is associated with the run for accounting purposes. The string can be used for generation of accounting reports.

- **-append**

  this is a switch for the **get** option. If the switch is present, then only new file content is retrieved and appended to the local file copy. Otherwise, the entire file is copied every time.

- **[-approval | -ap] [<job>][,<job>]...**

  approval jobs for the run. These jobs are scheduled first. After they complete, the run priority level is set to 10. The user needs to approve the run to return the priority level to its previous value. The run can be approved through the Eye. External tools can use the **run approve** API command to approve the run.

- **-completed**

  retrieves information about completed runs. This information is stored in the file **completed** in the **enfinfo** subdirectory of the current working directory.

- **[-count | -c] <number>**

  specify multiple jobs. This option can be used to execute the run multiple times. Jobs are distinguished by the environment variable ENFJOBNAME, which has a different value for each job. The option is used for command line programs or scripts. Run files already specify multiple jobs.

- **[-delete | -del]**

  delete a file from the EnFuzion root computer after it is fetched from the root computer to the local computer. By default, files are not deleted from the EnFuzion root computer. This option is used in conjunction with the **-fetch** option. If **-fetch** is not specified, then this option has no effect.

- **[-dir | -d] <path>[@<host_name>][,<path>[@<host_name>]]**

  specify the working job directory on nodes.

- **-e <user_name>@<host_name>,[<user_name>@<host_name>]**

  the list of recipients for e-mail notifications. Use the **-m** option to specify the condition for sending notifications.

- **[-export-environment | -x]**

  export the values of all environment variables from the submit host to the node.

- **-fail <number>**

  specify the maximum number of allowed failed jobs on a node. After <number> jobs fail on the node, no more jobs from the run are scheduled on the node.

- **[-fetch | -f]**

  fetch output files from the EnFuzion root computer. The output files are copied incrementally from the EnFuzion root computer to the submit computer as they are being created. This is useful for obtaining output files from completed jobs while there are still other jobs waiting or executing.

- **[-fetch-input | -fi]**

  fetch input files from the EnFuzion root computer. By default, only output files are being fetched. With this option, input files are being fetched as well. This option is used in conjunction with the **-fetch** option. If **-fetch** is not specified, then this option has no effect.

- **-get <file>**

  copy a file from the EnFuzion root computer to a local subdirectory. The file is copied to the **run-<runID>** subdirectory of the working directory.

- **-i <node_file>[=<submit_file>][,<node_file>[=<submit_file>]]**

  input files for the run. The files are first stored from the submit machine to the root machine and then made available to jobs on nodes.

- **[-localdir | -ldir] <directory>**

  change the default subdirectory for the **-rd** option. If **-rd** is not specified, then this command has no effect.

- **-login <file_name>**

  change the user identity to the one specified in the identity file **<file_name>**. The identity file is created with the **enfcmd identity** command.

- **-m [s | d | a | p | c]**

  the conditions to send e-mail notifications. **s** means execution start, **d** means execution done, **a** means execution abort, **p** means execution stop (pause), and **c** means execution approval (confirmation). Recipient addresses are specified with the **-e** option.

- **-max <number>**

  specify the maximum number of concurrently executing jobs for the run.

- **[-name | -n] <name>**

  the name of the run.

- **-nice [on|off][@<host_name>][,[on|off][@<host_name>]]**

  priority for execution of user jobs on nodes. A different option can be specified for different hosts. If nice is turned on, user jobs are executed at a background priority, allowing them to proceed only when the system would be otherwise idle.

  On Windows, nice executes processes at the IDLE_PRIORITY_CLASS class and THREAD_PRIORITY_ABOVE_NORMAL level. For example, a screen saver program on Windows is executed in same class but at a lower level THREAD_PRIORITY_NORMAL. On Linux/Unix, nice executes processes under the nice system call with the value of 10.

- **[-noautodetect | -nd]**

  disable automatic detection of input files. With this option, the parsing of the run file is disabled and only user specified files are copied.

  If this option is not specified and a run file is submitted, enfsub parses the tasks in the run file, identifies input files for the run and copies these input files from the submit computer to the EnFuzion root computer. These input files are copied in addition to any input files specified by the user on the

command line. If an input file is specified in the run file, but does not exist, then the file copy is not attempted.

- **-o <root_file>[=<node_file>][,<root_file>[=<node_file>]]**

  output files from the run. The files are copied from nodes and stored in the result directory on the root.

- **[-poll-delay | -pd] <seconds>**

  the delay in seconds between contacting the EnFuzion root. The default value is 60s. For some operations, such as checking for run completion or new file, The **enfsub** program periodically contacts the EnFuzion root. This option changes the default interval between contacts.

- **[-quiet | -q]**

  disable the fetch progress report on individual files. By default, **enfsub** prints out files that are being copied from the EnFuzion root computer to the local computer under the **-fetch** option. This option disables these messages.

- **-rd**

  wait for the run to complete and copy run results to a separate run directory on the local host. This option can be used to include **enfsub** in scripts that submit a run and then process its results. By default, the local directory is named **run-<runID>**. The default value can be changed with the **localdir** option.

- **-restart <number>**

  specify the number of times that a job can be rescheduled in the case of an error. When this number is reached, the job is terminated with an error.

- **[-results | -r]**

  wait for the run to complete and copy run results to the current working directory on the local host. This option can be used to include **enfsub** in scripts that submit a run and then process its results.

- **-root <host_name>:<port_number>**

  the address of the EnFuzion network service. The address can also be specified in the **submit.config** file. If the service address is not specified, a default value of **localhost:10102** is used.

- **[-start-time | -t] [[[[<year>-]<month>-]<day>-]<hour>:]<minutes>[.<seconds>]**

  specify the start time for the run execution. Run execution will be delayed until the start time.

- **-u <user_name>[@<host_name>][,<user_name>[@<host_name>]]**

specify user accounts for job execution on nodes.

- **[-value | -v] <name>[=<value>][,<name>[=<value>]]**

  specify environment variables and their values.

- **[-wait | -w]**

  wait for the run to complete. The **enfsub** program will not return until the run is completed. This option can be used to include **enfsub** in scripts that submit a run and then process its results.

- **[-wall-time | -wt] <hour>[:<minutes>[:<seconds>]]**

  the maximum wall time interval that the run is allowed to execute.

Enfsub uses the following API commands to submit a run:

```
enfcmd cluster add run name <run_name>
enfcmd copy <run_file> root:run-<run_id>
enfcmd copy <input_file_1> root:run-<run_id>
...
enfcmd copy <input_file_n> root:run-<run_id>

# the following two lines are used for parametric executions
enfcmd run <run_id> load <run_file>
enfcmd run <run_id> start

# the following line is used for command line programs and scripts
enfcmd run <run_id> add command <options>
```

The last few lines depend on whether the run is a parametric study or a command line program or a script.

The examples below illustrate how the same run can be performed using a command line program, a script or a run file.

Examples of Using the Enfsub Program

This section gives some examples of **enfsub** usage. It demonstrates the same run, but executed as a command line program, a script and a run file.

The following is the command line example for Windows:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cmd /c copy input.txt output.file"
```

The following is the same command line example for Linux/Unix:

```
enfsub -n sample -a myaccount \
    -i input.txt \
    -o output-\$ENFJOBNAME-\$ENFHOSTNAME.txt=output.file \
    -rd -count 2 -e user@domain.com -m d \
    "cp input.txt output.file"
```

The following is a Windows script that has the same effect as the command line example above:

```
@echo off
rem ENF -i script.bat
rem ENF -n sample -a myaccount
rem ENF -i input.txt
rem ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
rem ENF -rd -count 2 -e user@domain.com -m d

copy input.txt output.file
```

The script is submitted with:

```
enfsub script.bat
```

The following is a Linux/Unix script that has the same effect as the command line example above:

```
#!/bin/sh
#ENF -i script.sh
#ENF -n sample -a myaccount
#ENF -i input.txt
#ENF -o output-$ENFJOBNAME-$ENFHOSTNAME.txt=output.file
#ENF -rd -count 2 -e user@domain.com -m d

cp input.txt output.file
```

The script is submitted with:

```
enfsub ./script.sh
```

The following is the run file **sample.run** with the same results:

```
set ENFNOTIFY_ADDRESS "user@domain.com";
set ENFNOTIFY_CONDITION "done";
task main
    copy input.txt node:input.txt
    copy node:input.txt node:output.file
    copy node:output.file output-$ENFJOBNAME-$ENFHOSTNAME.txt
endtask
jobs
1
```

```
2
endjobs
```

The run is submitted from the command line as:

```
enfsub -n sample -a myaccount -rd sample.run
```

The Enfcmd Program

Enfcmd has the following syntax:

```
enfcmd [host <hostname> <port>] [refresh <seconds>] \
    show [detailed] [ cluster | run <run_id> | node <node_name> ] |
    submit <run_file> [ <input_file_1> ... <input_file_n> ] |
    copyrun <run_id>
    copy <file_name> user[:<directory>] |
    copy <file_name> root[:<directory>] |
    identity |
    <API_command>
```

The **host <hostname> <port>** command defines the host and the port of the Dispatcher that enfcmd connects to. The command is optional. If it is not specified, the values from the **submit.config** file are used.

If the **refresh** command is specified, enfcmd repeats the requested action every <seconds> seconds.

The **show** command prints out information about the Dispatcher. The following options can be added to the **show** command:

- **detailed**

  Detailed information is provided.

- **cluster**

  Only information about the cluster is printed out.

- **run <run name>**

  Only information about the <run_id> run is printed out. The **show <run_id>** command appends leading 0's to the <run_id>, if necessary. For example, <run_id> 11 will be expanded to 0000000011.

- **node <node_name>**

  Only information about the <node_name> node is printed out.

The **submit <run_file> [<input_file_1> ... <input_file_n>]** command submits a new run for execution. It performs the following steps:

- creates a new run,
- copies the run file <run_file> from the current directory on the submit system to the EnFuzion root system,
- copies input files <input_file_1> ... <input_file_n> from the current directory on the submit system to the run directory on the EnFuzion root system,
- and starts the run execution.

These steps are performed with the following API commands:

```
enfcmd cluster add run name <run_name>
enfcmd copy <run_file> root:run-<run_id>
enfcmd copy <input_file_1> root:run-<run_id>
...
enfcmd copy <input_file_n> root:run-<run_id>
enfcmd run <run_id> load <run_file>
enfcmd run <run_id> start
```

The **copyrun <run_id>** command copies files from the run directory on the EnFuzion root system to the current working directory on the local system. The run directory is preserved. The **copyrun <run_id>** command appends leading 0's to the <run_id>, if necessary. For example, <run_id> 11 will be expanded to 0000000011.

The **copy <file_name> user[:<directory>]** command copies file **<file_name>** from the EnFuzion root system to directory <directory> on the local system. If <directory> is omitted, the file is copied to the current working directory.

The **copy <file_name> root[:<directory>]** command copies file **<file_name>** from the local system to directory <directory> on the EnFuzion root system. If <directory> is omitted, the file is copied to the main Dispatcher directory.

The **identity** command generates a user identification file, named **<user>@<host_name>.enflogin**. **<user>** is the user account name on the submit system and **<host_name>** is the host name of the system. The file contains an encoded user identification string. The file can be copied to another system or user account to represent the same user.

<API_command> is used to pass API commands directly to the Dispatcher. **<API_command>** is an API command string.

The example below demonstrates the use of **enfcmd** to get all run identifiers from the Dispatcher:

Example:

```
# enfcmd host localhost 3521 cluster get runs
0237200033 0237200034
```

If no command is specified in the command line, the **enfcmd** reads the command from standard input, which is useful in scripts.

Using Enfcmd in a Script

The example below demonstrates how additional tasks can be specified by using the **enfcmd** in a shell.

Example:

```
#!/bin/sh
# parameter $1 is the port number of the Dispatcher
# parameter $2 is the run ID to which the script adds a new task

director="enfcmd host localhost $1"

echo $director

# Read task specification from stdin
$director <<EOF
    run $2 add task testtask
        execute echo testtask executed on ‘date‘
        node:execute ps -ef | grep enf > ps.$ENFJOBNAME
        node:copy node:ps.$ENFJOBNAME .
    endtask
EOF

# Add a job which executes the added task
# and return the status
addjob_status=‘$director run $2 add job task testtask‘

# Print status
echo Added task status:$addjob_status
```

The script accepts the Dispatcher port as the first argument and must be executed on the same host as the Dispatcher.

Handling of Privileges

Root options **privileges**, see details in the Section called *Enforcing Privileges* in Chapter 6, and **protect**, see details in the Section called *Prevent Execution of User Programs on the EnFuzion Root System* in Chapter 6, affect which actions can be performed by **enfsub** and **enfcmd** users. By default, **privileges** and **protect** are turned off, which allows any action to be performed by any user. The option **noanonsubmit**, see details in the Section called *Rejecting Anonymous Run Submission* in Chapter 6, does not affect **enfsub** and **enfcmd**, since they always provide a user ID with the job submission.

If **privileges** are turned on, then the following actions are permitted only by users with administrative privileges:

- enfcmd copy;
- if API commands are used by **enfcmd**, their usage follows the rules described in the Section called *Handling of Privileges* under the API commands.

If **privileges** are turned on, then the following actions are permitted only by users with administrative privileges and run owners:

- enfcmd copyrun;
- enfsub -attach;

If **protect** is turned on, then the user is not allowed to specify files outside of the run directory on the root. Files names must be relative and are not allowed to start with "/", "\", "<letter>:" or contain the string "..". The following commands are affected:

- enfcmd copy;
- enfcmd copyrun;
- enfcmd submit;
- enfsub -i;
- enfsub -o.

Access Control

The Dispatcher offers IP-based authentication for **enfsub** and **enfcmd** access. The administrator can set a list of IP addresses that are allowed or denied to connect to the Dispatcher with **enfsub** and **enfcmd** (see the Section called *Restricting Access to the Dispatcher Interface* in Chapter 6 for details).

HTTP Based Application Programming Interface

The section describes an HTTP based application programming interface (API), provided by the Dispatcher. The HTTP based API allows other applications to submit jobs to EnFuzion and retrieve results using HTTP, a standard Internet protocol. The HTTP protocol is an open protocol and HTTP libraries exist for most programming and scripting languages.

The HTTP based API is optimized for job submission and retrieval of results. It is different than the EnFuzion API, described in the Section called *Application Programming Interface*, which provides a more comprehensive set of monitoring and controlling operations.

The HTTP based interface must be enabled with the **httpport** option as described in the Section called *Port Number for the HTTP Based Interface* in Chapter 6. In the default EnFuzion configuration, the HTTP based interface is disabled, so it will not work without configuring **httpport**.

EnFuzion HTTP interface implements a number of requests. An external application can use these requests to submit jobs for execution and to obtain results. To demonstrate the use of the HTTP interface, EnFuzion distribution packages include a sample program implementation in the Python programming language. The program implements a subset of features of the EnFuzion **enfsub** command.

The sections below describe details about the HTTP based API, provide some examples of how the interface is being used and explain the Python **enfsub** example.

Description of HTTP Requests

The following sections provide detailed information about EnFuzion provided HTTP requests. EnFuzion keeps the network connection open after an HTTP request is completed, so multiple requests can be issued over the same connection. An open connection must be explicitly closed by the HTTP client.

Creating a New Run, POST newrun

The **newrun** command creates a new run. Its arguments are the run name, the owner user name and the account for the run. The arguments are optional. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the new run ID. Additional details on submitting a run for execution can be found in the Section called *Submitting Run for Execution*.

Request:

```
POST /cgi/newrun?runname=<run_name>&username=<user_name>&account=<account>
body: empty
```

Response:

```
status: 200, if OK
body: <run-ID>
```

Uploading a File, PUT

The **PUT** request uploads a file to the EnFuzion Dispatcher. Its argument is the target file path, starting with the run ID. The body of the request contains the file content. If the request is successful, the return status is 200 and the body is empty.

Request:

```
PUT <run-ID>/<file_name>
body: file content
```

Response:

```
status: 200, if OK
body: empty
```

Downloading a File, GET

The **GET** request retrieves a file from the EnFuzion Dispatcher. Its argument is the source file path, starting with the run ID. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the file content.

Request:

```
GET <run-ID>/<file_name>
body: empty
```

Response:

```
status: 200, if OK
body: file content
```

Deleting a File, POST deletefile

The **deletefile** command deletes a file from the EnFuzion Dispatcher. Its arguments are a run ID and the target file path. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the file was deleted. The file was deleted, if the body contains "1". Otherwise, there was an error in deleting the file.

Request:

```
POST /cgi/deletefile?runid=<run-ID>&filename=<file_name>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if deleted OK; 0, if error
```

Checking for File Existence, POST fileexists

The **fileexists** command checks for a file existence on the EnFuzion root. Its arguments are a run ID and the target file path. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the file exists. The file exists, the body contains "1". Otherwise, the body contains "0".

Request:

```
POST /cgi/fileexists?runid=<run-ID>&filename=<file_name>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if the file exists; 0, if the file does not exist
```

Starting a Run, POST startrun

The **startrun** command parses the run file, prepares the run for execution and submits the run to the Dispatcher. Its arguments are a run ID and the run file. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the run was submitted successfully. The submission was successful, if the body contains "1". Otherwise, there was an error in submitting the run. Additional details on submitting a run for execution can be found in the Section called *Submitting Run for Execution*.

Request:

```
POST /cgi/startrun?runid=<run-ID>&runfile=<file_name>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if started OK; 0, if error
```

Get All Files, POST getallfiles

The **getallfiles** command obtains the list of all files in the run directory on the EnFuzion root. Its argument is a run ID. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the list of files.

Request:

```
POST /cgi/getallfiles?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: a list of files, one per line
```

Get Input Files, POST getinputfiles

The **getinputfiles** command obtains the list of input files in the run directory on the EnFuzion root. Input files are files that were submitted before the run was started with the **startrun** request. The argument is a run ID. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the list of files.

Request:

```
POST /cgi/getinputfiles?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: a list of files, one per line
```

Get New Files, POST getnewfiles

The **getnewfiles** command obtains the list of new files in the run directory on the EnFuzion root. This request can be used to incrementally retrieve new files from the EnFuzion root while the jobs are still executing. The argument is a run ID. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the list of files. Additional details on incremental file retrieval can be found at the Section called *Incremental File Retrieval*.

Request:

```
POST /cgi/getnewfiles?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: a list of files, one per line
```

Get the Log File, POST getlogfile

The **getlogfile** command copies the run log from the EnFuzion root. The argument is a run ID. The body of the request is empty. If the request is successful, the return status is 200 and the body contains the run log file.

Request:

```
POST /cgi/getlogfile?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: file content
```

Set a File Copy Mark, POST setcopymark

The **setcopymark** command sets the new mark for incremental file copying. This request can be used to incrementally retrieve new files from the EnFuzion root while the jobs are still executing. The argument is a run ID. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the mark was set successfully. The mark was set successfully, if the body contains "1". Otherwise, there was an error in setting the mark. Additional details on incremental file retrieval can be found in the Section called *Incremental File Retrieval*.

Request:

```
POST /cgi/setcopymark?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if set OK; 0, if error
```

Checking for Run Start, POST runstarted

The **runstarted** command checks, if the run has been started. The argument is a run ID. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the run started. The run started and has been placed in the execution queue, if the body contains "1". Otherwise, the run was created, but not yet started.

Request:

```
POST /cgi/runstarted?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if run started; 0, otherwise
```

Checking for Run Completion, POST runcompleted

The **runcompleted** command checks, if the run completed. The argument is a run ID. The body of the request is empty. If the request has been processed, the return status is 200 and the body indicates, if the run completed. The run completed and all its jobs have been processed, if the body contains "1". Otherwise, the run is still executing or waiting for execution.

Request:

```
POST /cgi/runcompleted?runid=<run-ID>
body: empty
```

Response:

```
status: 200, if request is OK
body: 1, if run completed; 0, otherwise
```

Access Control

The Dispatcher offers IP-based authentication for access to the HTTP based interface. The administrator can set a list of IP addresses that are allowed or denied to connect to the HTTP interface (see the Section called *Restricting Access to the HTTP based Interface* in Chapter 6 for details).

Testing the HTTP Interface

The HTTP interface can be easily tested with a **telnet** client. Simply connect to the EnFuzion HTTP port as configured by the **httpport** root option and issue an HTTP request.

A sample HTTP session is shown below:

```
host3:/home/user1> telnet localhost 10108
Trying 127.0.0.1...
Connected to localhost.localnet.
Escape character is '^]'.
POST /cgi/newrun?runname=test&username=enfuzion&account=enfuzion HTTP/1.1
<press another Enter here>
HTTP/1.1 200 OK
Server: EnFuzion Submit Server 1.0
Date: Fri, 26 Nov 2004 00:11:15 GMT
Connection: Keep-Alive
Content-Type: text/plain
Content-Length: 10

0000000011
<press another Enter here>
Connection closed by foreign host.
```

This session demonstrates a telnet connection using the HTTP interface with a request that creates a new run. EnFuzion returns a request status and the ID of 0000000011 for the new run.

Submitting Run for Execution

Several HTTP requests are needed to submit a run. The process of submitting a run is as follows:

- the run is created with the **POST newrun** command;
- any input files are uploaded to the EnFuzion Dispatcher with the **PUT** request;
- the run is submitted for execution with the **POST startup** request.

More details on run submission using the HTTP interface is provided in the Section called *Submission with the HTTP Based Interface* in Chapter 9.

Incremental File Retrieval

HTTP requests can be used to incrementally retrieve files from the EnFuzion Dispatcher, while some jobs from the run are still executing.

The process of incremental file retrieval is as follows:

- get a list of new files with the **POST getnewfiles** command;
- download all the files from the list using the **GET** request;
- reset the copy mark with the **POST setcopymark** command;
- repeat the steps until the list is empty and the run completes. Run completion is tested with the **POST runcompleted** command.

More details on result retrieval using the HTTP interface is provided in the Section called *Retrieval with a Custom Program* in Chapter 9.

Implementation of Enfsub in Python

The EnFuzion distribution package includes a program in the Python programming language, which demonstrates the use of the HTTP based interface. The program implements a subset of features of the standard EnFuzion **enfsub** command. Although the Python implementation does not implement all the **enfsub** features, it has complete functionality to submit runs and retrieve the results.

The Python program consists of modules **enfuzion.py** and **enfsub.py**. The **enfuzion.py** module wraps EnFuzion HTTP based interface into high level Python functions. The module is generic and can be used in other applications. The **enfsub.py** implements **enfsub** specific functionality. It uses the **enfuzion.py** module to access the HTTP interface.

Both Python modules are provided as open source and can be modified and used in other applications, subject to the liability limitations in their license. The **enfsub.py** program requires that the EnFuzion root is installed and operational.

The Python program can be used as follows:

- download and install the Python programming language on your submit computer. Python is available for free from **www.python.org**. Some operating environments, such as Linux, usually have Python already installed;
- change your working directory to the EnFuzion **test** directory. On Linux/Unix, the default location is **$HOME/enfuzion/test**. On Windows, the default location is **C:\enfuzion\test**;
- copy **enfuzion.py** and **enfsub.py** from the EnFuzion **bin** directory to the current directory;
- submit the test **sample** run for execution:

  ```
  python enfsub.py -root <root_host>:<port> -pd 1 -fetch -run sample.run in-
  put.txt template.txt
  ```

  This command submits the sample run for execution, every second checks for output results and copies them to the local machine, and exits when the run execution is completed.

Application Programming Interface

This section describes the application programming interface (API), provided by the Dispatcher. This API is an EnFuzion specific network protocol that provides a wide range of functionality to control and monitor Dispatcher operation.

The EnFuzion API is different than the HTTP based API, described in the Section called *HTTP Based Application Programming Interface*. The HTTP based API is optimized for job submission and retrieval of results. The EnFuzion API is more comprehensive. Through the API, EnFuzion can be easily integrated with other programs into a seamless environment. External programs use the API to monitor and control EnFuzion operation by sending queries about the execution progress, by changing the configuration through changing variable values, and so forth.

The API has been designed to provide consistent syntax and ease of use. All API commands follow the same syntax: an object name, followed by a command and parameters. Object types are: cluster, connection, node, run, job and context. The commands that deal with object variables are generic, they work on all objects. These commands are **get**, **set** and **unset**. Objects have additional object specific commands.

The sections below describe how to connect to the Dispatcher. They give command descriptions and an example of how to use them from a program.

Connecting with the Dispatcher

An external program that uses the API to communicate with the Dispatcher is called a Director. The communication is handled via TCP/IP protocol. At the beginning of the execution, the Dispatcher prints

out the API port number in its log file. This port is the main communication channel between the cluster and other programs. The Director can connect to this port to establish a direct communication with the Dispatcher.

Format of Messages

The messages sent between the Dispatcher and a Director are variable size messages. A message consists of printable ASCII characters, and it must be terminated by a newline character '\n'.

Error Format

If a command is not valid, the Dispatcher returns an error message. All API error messages have the same syntax:

```
error <error_code> <error_text>
```

Example:

```
error 1 invalid command
```

Establishing a Connection

When a user connects to the Dispatcher API port, a connection is created. Multiple simultaneous connections can be created, one for each successful connection to the Dispatcher.

To connect with a program to the Dispatcher:

1. Connect to the API port on the Dispatcher.
2. Send command "director" or "director clu <user_identity>". <user_identity> is the string being produced with the **enfcmd identity** command. It follows the **clu** keyword in the ***.enflogin** file. If only "director" is being sent, then the owner user ID is set to the generic anonymous. Otherwise, the user ID is assigned based on <user_identity> as specified in the Section called *Specifying User Identities* in Chapter 6.
3. Get a reply from the Dispatcher.
4. If the reply is not "OK", disconnect and try later. If the authentication fails, the Dispatcher closes the connection without sending a reply.

By default, connections from any host are allowed. By setting the root option remoteaccess to false, only connections from the local host are allowed.

After the connection is established, API commands described in the following sections can be sent to the Dispatcher.

A connection has a type, which can be Director or Observer. Directors are used to send API commands to the Dispatcher and Observers receive cluster logs. By default, the connection is of type **director**. The connection type can be changed with commands **observe** and **direct**.

Direct

With command **direct**, the connection type is changed to a Director. A directing connection is used to send commands to the cluster. By sending the **observe** command to the connection, the directing connection's type can be changed to an Observer.

Return value: string "OK" if no errors or error message.

Observe

Command **observe** changes the connection type to an Observer. An observing connection receives cluster events. By sending the **direct** command to the connection, the observing connection's type can be changed back to a Director.

Return value: string "OK" if no errors or error message.

Description of Commands

Cluster Commands

cluster get

Obtain the value of a cluster variable:

```
cluster get [ <variable_name> ]
```

Return value: a string representing the variable value. If <variable_name> is omitted, all variable names are printed.

cluster set

Set a variable value:

```
cluster set <variable_name> <value>
```

Return value: string "OK" if no errors or error message. Some variables are read only and their value cannot be set.

cluster unset

Remove variable with the specified name:

```
cluster unset <variable_name>
```

Return value: string "OK" if no errors or error message. Some variables are required by the system and cannot be removed.

cluster start

Start cluster processes, cluster state is changed to ’running’:

```
cluster start
```

Return value: string "OK" if no errors or error message.

cluster abort

Terminate cluster execution, but does not exit the Dispatcher:

```
cluster abort
```

Return value: string "OK" if no errors or error message.

Abort executes the following actions:

- Terminate all nodes and jobs
- Stop all runs
- Terminate cluster root daemons
- Set cluster status to ’down’

cluster shutdown

Terminate cluster execution and the Dispatcher process:

```
cluster shutdown
```

Return value: string "OK" if no errors or error message.

cluster add run

Add a run to the cluster:

```
cluster add run file <file_name> [ directory <directory> ] [ account <account> ]
cluster add run name <run_name> [ directory <directory> ] [ account <account> ]
```

Return value: **<run_id>** of the newly created run.

Command options are:

- **file <file_name>**

  The run information is read from the run file <file_name>. The run is named by using the file prefix.

- **name <run_name>**

  An empty run is created with the provided name.

- **directory <directory>**

  Specifies a directory which is used as a working directory for the run. If a directory is not specified, it is automatically created.

- **account <account>**

  Specifies an account string which can be used later for filtering account information.

cluster remove run

Remove a run from the cluster.

```
cluster remove run <run_id>
```

Return value: string "OK" if no errors or error message. Any executing jobs from the run are terminated.

cluster add node

Add a new node to the cluster:

```
cluster add node <host_name> <user_name> <password> [<type>]
```

Return value: **<node_id>** of the newly created node.

Command options are:

- **<host_name>**

  The name of the host where node executes

- **<user_name>**

  Username used to execute the node

- **<password>**

  Username password on the node host

- **<type>**

  Optional node type, one of Unix, WindowsNT, UnixRsh, ssh, command

cluster remove node

Remove a node from the cluster:

```
cluster remove node <node_id>
```

Return value: string "OK" if no errors or error message.

Node Commands

node get

Obtain the value of a node variable:

```
node <node_id> get [ <variable_name> ]
```

Return value: a string representing a variable value. If <variable_name> is omitted, all variable names are printed.

node set

Set a variable value:

```
node <node_id> set <variable_name> <value>
```

Return value: string "OK" if no errors or error message. Some variables are read only and their value cannot be set.

node unset

Remove a variable with the specified name:

```
node <node_id> unset <variable_name>
```

Return value: string "OK" if no errors or error message. Some variables are required by the system and cannot be removed.

node start

Initiate node activation:

```
node <node_id> start
```

Return value: string "OK" if no errors or error message. The node is set to state ’starting’ until it is activated. After it is activated, the node state changes to ’ready’.

node terminate

Initiate node termination:

```
node <node_id> terminate
```

Return value: string "OK" if no errors or error message. The node is in state ’terminating’ until it is terminated. When the node terminates, its state changes to ’down’.

Run Commands

run get

Obtain the value of a run variable:

```
run <run_id> get [ <variable_name> ]
```

Return value: a string representing a variable value. If <variable_name> is omitted, all variable names are printed.

run set

Set a variable value:

```
run <run_id> set <variable_name> <value>
```

Return value: string "OK" if no errors or error message. Some variables are read only and their value cannot be set.

run unset

Remove a variable with the specified name:

```
run <run_id> unset <variable_name>
```

Return value: string "OK" if no errors or error message. Some variables are required by the system and cannot be removed.

run start

Start run execution:

```
run <run_id> start
```

Return value: string "OK" if no errors or error message. The state of the run is changed to ’executing’.

run stop

Stop run execution:

```
run <run_id> stop
```

Return value: string "OK" if no errors or error message. Executing jobs are rescheduled and no new jobs are started from the run. Job execution can be restarted with the **run start**. Run directory is preserved.

run abort

Abort run execution by the user.

```
run <run_id> abort
```

Return value: string "OK" if no errors or error message. Execution of all run jobs is terminated and the run is removed from the Dispatcher. The run cannot be restarted. Run directory is preserved.

run approve

Approve continued run execution.

```
run <run_id> approve
```

Return value: string "OK" if no errors or error message. The run is approved for continued execution. The original run priority level is restored.

run reschedule

Reschedule failed jobs from the run. by the user.

```
run <run_id> reschedule
```

Return value: string "OK" if no errors or error message. All failed jobs from the run are rescheduled for another execution.

This API command works also for runs that have already completed. Completed runs are restarted and only their failed jobs are submitted for execution.

run load

Add definition from a run file to an existing run:

```
run <run_id> load <run_file>
```

Return value: string "OK" if no errors or error message. Defines additional jobs, tasks and parameters as specified in the run file.

run add command

Add definitions to an existing run and start the run:

```
run <run_id> add command
    nice on | off
    env <name> "<value>"
    input "<root_file>" "<node_file>"
    execute "<command>"
    output "<node_file>" "<root_file>"
    wall <seconds>
    email "<address>"
    wait
    esend start | done | abort
    results
    start [[[[<year>-]<month>-]<day>-]<hour>:]<minutes>[.<seconds>]
    user "<address>"
    directory "<path>"
    max <integer>
    count <integer>
```

Return value: string "OK" if no errors or error message.

This command is used primarily for run submission by enfsub. After receiving the command, the Dispatcher sets run account, node user, node directory, execution limit, e-mail conditions and recipients, creates jobs, sets job environment variables and starts the run.

Internally, a run file is generated by the Dispatcher with the following tasks:

```
task main
    copy <root_file> node:<node_file>
    ... # copy additional input files
    node:execute <command>
    ... # execute additional commands
    copy node:<node_file> <root_file>
    ... # copy additional output files
endtask
```

run add job

Add a job to the run:

```
run <run_id> add job \
    [ name <job_name> ] \
    [ task <task_name> ] \
    [ host <host_name> ]
    [ node <node_id> ] \
    [ parameters <count> [ <par_name> <value> ] ... ]
```

Return value: the command returns a string with the job name, if successful. Otherwise, it returns an error message.

Options to the command are:

- **<job_name>**

  the name of the job. Default is "j<number>", where the number is uniquely assigned by the Dispatcher.

- **<task_name>**

  the name of the main job task. Default is "main".

- **<host_name>**

  the host name of the node to execute the job. If the node with this host name is not defined, an error message is returned.

- **<node_id>**

  the node ID of the node to execute the job. If the node with this ID is not defined, an error message is returned.

- **<count>**

  the number of job parameters. This is the number of <par_name>, <value> pairs that follow.

- **<par_name>**

  parameter name

- **<value>**

  parameter value. A string in double quotes "...".

run add task

Add a task to the run:

```
run <run_id> add task <task_name>
```

Return value: string "OK" if no errors or error message.

This command reads lines of text until a line with "endtask" is encountered. All lines between "task" and "endtask" are part of this task.

Example using a Unix Bourne-shell script:

```
echo Adding main task
```

```
enfcmd <<EOF
    run 0001100002 add task main
        node:execute echo build task executed on ‘date‘ > build.out
        copy node:build.out root:build.$ENFJOBNAME
        node:execute sleep 10
    endtask
EOF
```

run usein datafile

Read datajob inputs directly from the file:

```
run <run_id> usein datafile <file_name>
```

Return value: string "OK" if no errors or error message.

run useout datafile

Store datajob outputs directly to the file.

```
run <run_id> useout datafile <file_name>
```

Return value: string "OK" if no errors or error message.

run in data

Submit one datajob input for execution. Input must be included in quotes:

```
run <run_id> in data "<input>"
```

Return value: string "OK" if no errors or error message.

run out data

Return the next datajob result:

```
run <run_id> out data
```

Return value: string "OK" if no errors or error message.

The command returns the next datajob result, starting with the ’data’ keyword. If no outputs are available, string ’nodata’ is returned. If datajobs completed and no outputs are available, string ’EOS’ is returned.

run poll data

Check out if the next datajob result is available:

```
run <run_id> poll data
```

Returns 0, if no outputs are available. Otherwise, it returns a positive number.

run movein datafile

Submit a file with datajob inputs for execution:

```
run <run_id> movein datafile <file_name>
```

Return value: string "OK" if no errors or error message. The command moves the file if it is on the same partition as the Dispatcher directory. Otherwise, it copies the file. This command removes the file.

run copyin datafile

Submit a file with datajob inputs for execution:

```
run <run_id> copyin datafile <file_name>
```

Return value: string "OK" if no errors or error message. The command copies the file to the Dispatcher directory. It does not change the original file.

run moveout datafile

Store datajob output to the file:

```
run <run_id> moveout datafile <file_name>
```

Return value: string "OK" if no errors or error message.

Job Commands

job get

Obtain the value of a job variable:

```
job <run_id> <job_name> get [ <variable_name> ]
```

Return value: a string representing a variable value. If <variable_name> is omitted, all variable names are printed.

job set

Set a variable value:

```
job <run_id> <job_name> set <variable_name> <value>
```

Return value: string "OK" if no errors or error message. Some variables are read only and their value cannot be set.

job unset

Remove variable with the specified name:

```
job <run_id> <job_name> unset <variable_name>
```

Return value: string "OK" if no errors or error message. Some variables are required by the system and cannot be removed.

job abort

Abort job execution by the user.

```
job <run_id> <job_name> abort
```

Return value: string "OK" if no errors or error message. Job execution is aborted, and job status is changed to ’failed’. If a job has not yet been executed, the job is removed from the ready queue of the run. If a job has completed, its status is changed to failed.

job reschedule

Reschedule job execution:

```
job <run_id> <job_name> reschedule
```

Return value: string "OK" if no errors or error message. If the job is executing, it is terminated, and rescheduled. Its status changes to ’ready’. If a job already completed or failed, it is scheduled for another execution.

Context Commands

context set property

Add a property to a context:

```
run <run name> set -context <node name> ENFCONTEXT_PROPERTIES <prop>
```

Return value: string "OK" if no errors or error message. Context properties are valid for a node only within a run.

context unset property

Remove a property from a context:

```
run <run name> unset -context <node name> ENFCONTEXT_PROPERTIES <prop>
```

Return value: string "OK" if no errors or error message. Context properties are valid for a node only within a run.

Connection Commands

connection get

Obtain the value of a connection variable:

```
connection get [ <variable_name> ]
```

Return value: a string representing the variable value. If <variable_name> is omitted, all variable names are printed.

connection get admin

Verify administrative privileges:

```
connection get admin
```

Return value: OK, if the caller has administrative privileges. Otherwise, return an error.

connection close

Close the connection to the Dispatcher:

```
connection close
```

Return value: string "OK" if no errors or error message.

Handling of Privileges

Root options noanonsubmit, see details in the Section called *Rejecting Anonymous Run Submission* in Chapter 6, and privileges, see details in the Section called *Enforcing Privileges* in Chapter 6, affect which API commands can be performed by users. By default, noanonsubmit and privileges are turned off, which allows any API command to be performed by any user.

If noanonsubmit is turned on, then the following API command is not permitted by users with the anonymous user ID:

- **cluster add run**.

If privileges are turned on, then only the following API commands are permitted by users without administrative privileges, administrators have no limitations:

- **cluster get** can be executed by any user;
- **cluster add run** can be executed by the run owner;
- **cluster remove run** can be executed by the run owner;
- **node get** can be executed by any user;
- **run get** can be executed by the run owner;
- **job get** can be executed by the run owner;
- **context get** can be executed by the run owner.

Access Control

The Dispatcher offers IP-based authentication for access to the programming interface. The administrator can set a list of IP addresses that are allowed or denied to connect to the Dispatcher programming interface (see the Section called *Restricting Access to the Dispatcher Interface* in Chapter 6 for details).

Using the Programming Interface From C

The program Enfdirector provides an example of the use of the directing protocol. The source code demonstrates all the necessary steps to connect to the Dispatcher port and send commands specified by the protocol. If required, the code can be easily modified so that it sends several commands before disconnecting.

Example:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>

#define LINELEN 1024

/* Function declarations */

int directorlogin(char *host, int dispport);
long int gethostaddr(char *name);
```

```
int socconnect(long int addr, int port);

main(int argc, char **argv)
{
    int tn;
    char command[LINELEN];
    char *hostname;
    int hostport;
    int argpos;
    char ch;

    if (argc < 3) {
        fprintf(stderr, "Usage: %s hostname port [command]\n",argv[0]);
        exit(1);
    }
    argpos = 1;
    hostname = argv[argpos];
    argpos++;
    if (sscanf(argv[argpos], "%d", &hostport) != 1) {
        fprintf(stderr, "Invalid port\n");
        exit(1);
    }
    argpos++;
    strcpy(command,"");
    if (argpos < argc) {
        /* Concatenate arguments */
        while (argpos < argc) {
            strcat(command, argv[argpos]);
            argpos++ ;
            if (argpos < argc) {
                strcat(command, " ");
            }
        }
        strcat(command, "\n");
    }
    tn = directorlogin(hostname,hostport);
    if (tn < 0) {
        fprintf(stderr, "*** Error: director login failed\n");
        exit(1);
    }
    if (strcmp(command,"") == 0) {
        /* No commands in arguments, read from stdin */
        while (fgets(command,LINELEN,stdin) != NULL) {
            write(tn, command, strlen(command));
        }
    } else {
        write(tn, command, strlen(command));
    }
    while(read(tn,&ch,1) > 0) {
        printf("%c", ch);
        if(ch == '\n') break ;
    }
    close(tn);
```

```
}

/* Connect as director */
int directorlogin(char *hostname, int dispport)
{
    long int iaddr;
    long int hostaddr;
    int sd;
    unsigned char buf[1024];
    char *str;

    hostaddr = gethostaddr(hostname);
    if (hostaddr == -1) {
        fprintf(stderr, "Unable to get address of host %s\n", hostname);
        return -1;
    }
    iaddr = ntohl(hostaddr);
    if (iaddr == -1) {
        fprintf(stderr, "Unable to convert host address to host order\n");
        return(-1);
    }
    sd = socconnect( iaddr, dispport );
    if (sd < 0) {
        fprintf(stderr, "Unable to connect to host %x port %d\n", iaddr, dispport);
        return(-1);
    }
    str = "director\n" ;
    if (write(sd,str,strlen(str) ) == -1) {
        fprintf(stderr, "Unable to write 'director'\n");
        return -1;
    }
    if (read(sd, buf, 3) == -1) {
        fprintf(stderr, "Unable to read 'director' response\n");
        return -1;
    }
    if (strcmp(buf, "OK\n") != 0) {
        fprintf(stderr, "Invalid response for 'director'\n");
        return -1;
    }
    return sd;
}

/* Get IP address of host */
long int gethostaddr(char *name)
{
    unsigned char ad[4];
    int i;
    struct hostent *host;
    char host_ad[50];
    host = gethostbyname(name);
    if (host == (struct hostent *)0) {
        return(-1);
    }
```

```
    for(i = 0; i < 4; i++)  ad[i] = host->h_addr[i];
    sprintf(host_ad, "%u.%u.%u.%u", ad[0], ad[1], ad[2], ad[3]);
    return(inet_addr(host_ad));
}

/* connect to a socket */
/* addr: host Internet address */
/* port: port address */
/* return the new socket descriptor, if successful */
/* return -1, otherwise */
int socconnect(long int addr, int port)
{
    struct sockaddr_in scket;
    int sd;

    sd = socket(PF_INET,SOCK_STREAM,0);
    if (sd < 0) {
        return(-1);
    }
    memset (&scket,0,sizeof(scket));
    scket.sin_family = PF_INET;
    scket.sin_port = htons((u_short) port);
    scket.sin_addr.s_addr = htonl(addr);
    if (connect(sd,(struct sockaddr *) &scket,sizeof(scket)) < 0) {
        close(sd);
        return (-1);
    }
    return(sd);
}
```

Chapter 11. Program Reference

This chapter provides a reference for the various EnFuzion programs. A short description of each program is provided, including its function, its options and references to further details about its use.

Enfacct

The Enfacct program extracts the accounting data from log files. The program scans the main EnFuzion log file, run log files and the enfinfo directories and stores the accounting data to the enfinfo/acct directory. The accounting data is generated every hour. The Enfacct is executed automatically by the Dispatcher within first 5 minutes of every hour.

The Enfacct program has the following options:

```
enfacct \
    [ -verbose [ <file> ] ]  \
    [ -dir <directory> ] \
    [ -strict ] \
    [ -complete ] \
    [ -aggregate [ YYYY-MM-DD ] ]
```

The **-verbose [<file>]** option turns on verbose output. The option is useful for troubleshooting. If <file> is specified, the output is saved in the file. Otherwise, it is sent to the standard output.

-dir <directory> specifies the main Dispatcher directory. The option can be used when enfacct is started in a directory that is different from the main Dispatcher directory.

With the **-strict** option, Enfacct terminates if a parsing error is encountered. A parsing error might happen if the log files are modified by the user by mistake. The option is useful for troubleshooting.

With the **-complete** option, Enfacct does a complete parsing of log files. Any partial results are discarded. During normal operation, Enfacct performs incremental parsing of log files. The log parsing continues where the previous instance of Enfacct stopped.

With the **-aggregate [YYYY-MM-DD]** option, Enfacct does not extract new accounting data from logs, but it aggregates existing accounting data. Daily summaries are produced from hourly summaries and monthly summaries are produced from daily summaries.

Enfacct automatically deletes any hourly files that are more than 2 days old and any daily files that are more than 2 months old.

The accounting data is stored in the **enfinfo/acct** subdirectory in the working Dispatcher directory. There are two files for each time period. **runs-*** files contain accounting information about runs and **nodes-*** files contain accounting information about nodes.

Files for each completed hour of the current and previous day are kept in:

```
<year>-<month>-<day>/runs-<year>-<month>-<day>-<hour>
<year>-<month>-<day>/nodes-<year>-<month>-<day>-<hour>
```

Files for each completed day of the current and previous month are kept in:

```
<year>-<month>/runs-<year>-<month>-<day>
<year>-<month>/nodes-<year>-<month>-<day>
```

Files for each completed month are kept in:

```
<year>/runs-<year>-<month>
<year>/nodes-<year>-<month>
```

Enfcmd

The Enfcmd utility is used to communicate with the Dispatcher. The Enfcmd program supports most common tasks with simple commands. It also implements a complete Dispatcher API, so that API commands can be easily used with scripts.

The Enfcmd program uses the following syntax:

```
enfcmd [host <hostname> <port>] [refresh <seconds>] \
    show [detailed] [ (cluster | run <run_id> | node <node_name>) ] |
    submit <run_file> [ <input_file_1> ... <input_file_n> ] |
    copyrun <run_id> |
    copy <file_name> user[:<directory>] |
    copy <file_name> root[:<directory>] |
    identity |
    <API_command>
```

The **host <hostname> <port>** command defines the host and the port of the Dispatcher that Enfcmd connects to. The command is optional. If it is not specified, the values from the **submit.config** file are used.

If the **refresh** command is specified, Enfcmd repeats the requested action every <seconds> seconds.

The **show** command prints out information about the Dispatcher or its individual components. The **show <run_id>** command appends leading 0's to the <run_id>, if necessary. For example, <run_id> 11 will be expanded to 0000000011.

The **submit <run_file> [<input_file_1> ... <input_file_n>]** command submits a new run for execution.

The **copyrun <run_id>** command copies files from the run directory on the EnFuzion root system to the current working directory on the local system. The **copyrun <run_id>** command appends leading 0's to the <run_id>, if necessary. For example, <run_id> 11 will be expanded to 0000000011.

The **copy <file_name> user[:<directory>]** command copies file **<file_name>** from the EnFuzion root system to directory <directory> on the local system.

The **copy <file_name> root[:<directory>]** command copies file **<file_name>** from the local system to directory <directory> on the EnFuzion root system.

The **identity** command generates a user identification file, named **<user>@<host_name>.enflogin**. **<user>** is the user account name on the submit system and **<host_name>** is the host name of the system.

The file contains an encoded user identification string. The file can be copied to another system or user account to represent the same user.

<API_command> is used to pass API commands directly to the Dispatcher. **<API_command>** is an API command string.

More details about **enfcmd** are available in the Section called *The Enfcmd Program* in Chapter 10.

Enfdispatcher

The Dispatcher is the main program on the EnFuzion root system, controlling job execution and other EnFuzion processes. It can be used to process a single run as a command line utility, or multiple runs as a server program.

The Dispatcher can be started from the command line as:

```
enfdispatcher   [ options ]   [<run_file>]
```

The Dispatcher reads its options and takes an optional run file. The optional run file is useful to provide the run description in a command line, when the Dispatcher is executed in a single run mode.

The Dispatcher command line options are:

- **-help**

  If this is the first option, then the Dispatcher prints out a help notice and exits. If it is not the first option, then **-help** has no effect.

- **-d**

  The Dispatcher is placed in a daemon mode . On Linux/Unix systems, the Dispatcher performs the following steps: forks twice, becomes a session leader, and closes the standard file descriptors. On Windows, the Dispatcher calls itself with its original command line arguments, except for the "-d" argument, which is removed. The new process shares the same working directory, but is in a new process group, has a new console, which is not shown on the screen and does not inherit the handles. The original Dispatcher exits.

- **-m**

  The Dispatcher is executed in a multi run mode. By default, the Dispatcher is executed in a single run mode, where it executes one run either specified on a command line or a previously interrupted run and exits. In the multi run mode, the Dispatcher continuously processes runs until it is terminated by the administrator or by the system. The multi run mode is useful to provide EnFuzion as a network service.

- **-p <port_number>**

This option changes the default port number of its network based application programming interface to **<port_number>**. By default, the Dispatcher uses port 10102. The application programming interface is described in the Section called *Application Programming Interface* in Chapter 10.

- **-r**

  This option recovers uncompleted runs from a previous Dispatcher. If the EnFuzion root system fails or the Dispatcher is terminated, then some of the runs might not be completed. If a new Dispatcher is restarted with the **-r** option in the same directory as the terminated Dispatcher, then it will reload the uncompleted runs and execute them to completion.

- **-v**

  If this is the first option, then the Dispatcher prints out its version and exits. If it is not the first option, then **-v** has no effect.

- **-w <directory>**

  The Dispatcher sets its working directory to the **<directory>** path. The working directory contains the Dispatcher log files and other working files.

  This option is useful for safely setting the working directory, for example when the Dispatcher is executed using a scripting language or from a Java class.

- **<run_file>**

  This specifies the run file to process in a single run mode . Single run mode is suitable for executing the Dispatcher in scripts and from a command line. In single run mode, the Dispatcher takes a run file as input, automatically starts processing the jobs and exits after all the jobs complete. If all the jobs complete successfully, the Dispatcher returns 0 as its exit value. If some of the jobs fail, the Dispatcher returns 1 as its exit value.

  In single run mode, nodes are usually provided in the file enfuzion.nodes before the execution starts.

Most of the root options, described in the Section called *Specifying Root Configuration Options* in Chapter 6, can also be specified on the command line. The command line value takes precedence over the value in the **root.options** file. The root options that can be specified from the command line are:

- **-bind**, which determines if nodes can operate in the autonomous mode. See the Section called *Autonomous Node Operation* in Chapter 6 for details.
- **-cleanuplimit**, which specifies the period to delete the obsolete user directories. See the Section called *Deleting Obsolete User Directories* in Chapter 6 for details.
- **-commport**, which specifies the port to broadcast the root host and port on the local network. See the Section called *Port Number for Broadcasting the Address* in Chapter 6 for details.

- **-completelogs**, which turns on run specific events in the main cluster log. See the Section called *Complete Logs* in Chapter 6 for details.
- **-disconnect**, which specifies the period that either a root or a node machine waits for a heartbeat signal. See the Section called *Disconnect Period* in Chapter 6 for details.
- **-eyeport**, which specifies the Eye port number. See the Section called *Port Number for the Eye* in Chapter 6 for details.
- **-eyestart**, which specifies, if the Eye is automatically started by the Dispatcher. See the Section called *Starting the Eye* in Chapter 6 for details.
- **-eyeterminate**, which specifies, if the Eye is terminated by the Dispatcher. See the Section called *Terminating the Eye* in Chapter 6 for details.
- **-heartbeat**, which specifies the interval for heartbeat between the root and the node machines. See the Section called *Heartbeat Period* in Chapter 6 for details.
- **-httpport**, which specifies the port number for the HTTP based interface. See the Section called *Port Number for the HTTP Based Interface* in Chapter 6 for details.
- **-jobport**, which specifies the port number that is used by user jobs on EnFuzion nodes to execute services on the root. See the Section called *Port Number for Job Execution* in Chapter 6 for details.
- **-logsizelimit**, which limits the size of the Dispatcher log for log rotation. See the Section called *Maximum Dispatcher Log Size* in Chapter 6 for details.
- **-mailport**, which specifies port of the SMTP service host for electronic notification messages. See the Section called *Specifying Mail Service Port* in Chapter 6 for details.
- **-mailserver**, which specifies the SMTP server host for electronic notification messages. See the Section called *Specifying Mail Server System* in Chapter 6 for details.
- **-mailuser**, which specifies the sender for electronic notification messages. See the Section called *Specifying Mail Sender* in Chapter 6 for details.
- **-maxdatastream**, which specifies the maximum size for a datajob. See the Section called *Maximum Datastream Job Size* in Chapter 6 for details.
- **-maxstart**, which limits the number of concurrent node activations. See the Section called *Concurrent Node Activations* in Chapter 6 for details.
- **-multinodes**, which allows multiple nodes on a single computer. See the Section called *Multiple Remote Nodes from One Host* in Chapter 6 for details.
- **-noanonsubmit**, which denies run submission by users with the anonymous ID. See the Section called *Rejecting Anonymous Run Submission* in Chapter 6 for details.
- **-privileges**, which enforces user privileges. See the Section called *Enforcing Privileges* in Chapter 6 for details.
- **-protect**, which denies execution of user programs on the root system. See the Section called *Prevent Execution of User Programs on the EnFuzion Root System* in Chapter 6 for details.
- **-restart**, which specifies the node restart period. See the Section called *Node Restart Period* in Chapter 6 for details.
- **-rootport**, which specifies the port that is used by nodes to connect to the root when they are started independently. See the Section called *Port Number for Node Connections* in Chapter 6 for details.

- **-remoteaccess**, which denies remote access to the Dispatcher API port. See the Section called *Allowing Remote Access to the Dispatcher Interface* in Chapter 6 for details.
- **-resources**, which specifies how often nodes should report their resource usage. See the Section called *Minimum Time to Obtain Resource Information* in Chapter 6 for details.
- **-queue**, which turns on the queuing policy for scheduling. See the Section called *Queueing Policy* in Chapter 6 for details.
- **-startport**, which specifies the port that the enfnodestarter program uses to accept node requests during the node start sequence. See the Section called *Port Number for Node Starter Connections* in Chapter 6 for details.
- **-waitlimit**, which limits the time that nodes can operate in the autonomous mode. See the Section called *Wait Limit* in Chapter 6 for details.

More details about the Dispatcher are available in the Section called *The Dispatcher* in Chapter 9.

Enfexecute

The program Enfexecute takes a task command as a command line option and executes that command. See the Section called *Task Commands* in Chapter 8.

```
enfexecute 'task_command'
```

The **enfexecute** command can be called from any program or scripting language.

More details about **enfexecute** are available in the Section called *Program Enfexecute* in Chapter 8.

Eye

The Eye program provides a web based interface, so that standard web browsers can be used to communicate with EnFuzion. Normally, the Eye's starting and termination are handled by the Dispatcher.

The Eye can be started from the command line as:

```
enfeye \
    [ -v ] \
    [ -auto-config ] \
    [ -requires-dispatcher ] \
    [ -dispatcher-port <port> ] \
    [ -http-port <port> ] \
    [ -root-dir <directory> ] \
    [ -tmp-dir <directory> ] \
    [ -static-html-dir <directory> ]
```

A description of options:

- **-v**

  Prints out the version number and exits.

- **-auto-config**

  Directories for EnFuzion logs and runs are acquired from the Dispatcher on each reconnection.

- **-requires-dispatcher**

  The Eye terminates if it cannot connect to the Dispatcher.

- **-dispatcher-port <port>**

  The Dispatcher port that the Eye connects to. The default value is retrieved from the EnFuzion log file.

- **-http-port <port>**

  The port that the Eye clients connect to with a web browser. The default value is 10101.

- **-root-dir <directory>**

  The working directory of the Dispatcher. Defaults to the current working directory. This option is ignored, when the **-auto-config** option is specified, since the directory is retrieved from the Dispatcher.

- **-tmp-dir <directory>**

  The directory for temporary files generated by the Eye. It defaults to the EnFuzion temporary directory. This directory must exist and must writable by the Eye.

- **-static-html-dir <directory>**

  Directory for storing static HTML files. The default value is the **html** subdirectory in the EnFuzion installation directory.

More details about the use of the Eye are available in the Section called *Graphical Web Based Interface* in Chapter 10.

Enfgenerator

The Generator takes a plan file, containing job templates and a description of parameters. It produces an application specific, graphical user interface, which is used to select parameter values. After the parameters values are selected, the Generator produces a run file, which contains a complete description of jobs and parameter values for each job.

The Generator can be started on a command line as:

```
enfgenerator [ -g ] [ <plan_name> ]
```

If the option -g is specified on the command line, the Generator will be executed with no graphical interface in batch mode. This mode is useful for calling the Generator directly from other programs.

More details about the Generator are available in the Section called *The Generator* in Chapter 8.

Enfinstall

The **enfinstall** command can be used to install EnFuzion on remote systems, without any need to access the system's keyboard or monitor. The program can also be used to install an EnFuzion license, verify an EnFuzion configuration and copy the options file to the nodes.

The Enfinstall program is called with a command option:

```
enfinstall <command>
```

The commands are:

- **enfuzion**

  Installs EnFuzion node software on node systems. /usr/local/enfuzion.

- **license**

  Installs an EnFuzion license to node systems.

- **verify**

  Accesses nodes and verifies their installation.

- **options**

  Copies the **enfuzion.options** file to node systems.

- **collect**

  Collects the information about EnFuzion nodes.

More details about **enfinstall** are available in the Section called *Enfinstall Program* in Chapter 4.

Enfkey

The Enfkey utility is used to perform basic tasks, such as generating new keys, and adding and removing keys. If a user defined authentication library is provided, **enfkey** uses that library.

The Enfkey program uses the following syntax:

```
enfkey keygen
```

This generates new public and private keys for the system where enfkey is executed. The IP address of the system that generated the keys is also printed to the standard output.

Example:

For the default EnFuzion authentication library, the keys are placed in file **enf_key.priv**. A sample file contents is:

```
Id=172.12.85.23
PrivKey=11773C2ADB11EBE6FBE7911056C3A1E53A4C7F4B
PublicKey=97F035A7B89B95CBA91F3EE1E3343293CACDDECD59D7
CA381490532BB118ECD204703702137E80CFB89EA622CE153699DE
2060CDB787A153B6321CFC376C7C97913D3C1795015A10FC3C9935
236DD68C2C3BC11E9142787600361F1AEF9EC9B82137270E1F175A
A1F52836030776AE0DA6FE5E4CB5E1C16C0EC60058DC0F47F1
```

Id designates the IP address of the system where the keys were created. PrivKey and PublicKey contain private and public keys, respectively.

Additional details about **enfkey** and its use are available in the Section called *The Enfkey Utility* in Chapter 7.

Enfkill

The Enfkill utility provides an emergency termination of EnFuzion nodes. The program causes all EnFuzion nodes to clean up their workspace files and directories and to terminate any EnFuzion activity on nodes. Enfkill is supported only on Windows NT/2000/XP platforms.

Enfkill has no command line options. It is executed on the root system by:

```
enfkill
```

Enfkill retrieves nodes from the **enfuzion.nodes** file in its working directory. If there is not **enfuzion.nodes** file in the working directory, enfkill takes the file from the EnFuzion configuration directory. The default path is **C:\enfuzion\config\enfuzion.nodes**. For each node, it terminates all EnFuzion user tasks and deletes the EnFuzion temporary files (see the Section called *The Enfkill Utility* in Chapter 3).

Enfmail

The Enfmail utility sends electronic messages. On Linux/Unix systems, it uses the local mail program by default. If an SMTP server is specified, which is required on Windows, it is used by the enfmail to send messages.

Enfmail has the following options:

```
enfmail \
    [ -server <SMPT_server_name> ] \
    [ -port <port> ] \
    [ -l <sender_address> ] \
    [ -t <address>(<description>)[,<address>(<description>) ] \
    [ -s <subject> ] \
    -body <file_name>
```

A description of options:

- **-server <SMPT_server_name>**

  Specify the SMTP server host.

- **-port <port>**

  Specify the SMTP server port.

- **-l <sender_address>**

  Specify the sender address.

- **-t <address>(<description>)[,<address>(<description>)**

  Specify the list of recipients and their descriptions.

- **-s <subject>**

  Specify the message subject.

- **-body <file_name>**

  Specify the file with the message text.

The program is used internally by the Dispatcher to send electronic notifications. Some of the **enfmail** parameters might need to be configured through root options as described in the Section called *Specifying Mail Server System* in Chapter 6, the Section called *Specifying Mail Sender* in Chapter 6, and the Section called *Specifying Mail Service Port* in Chapter 6.

Enfmail can be executed manually from a command line. This can be useful for verification of the e-mail sending mechanism during troubleshooting of EnFuzion notifications. A sample command to test enfmail is:

```
enfmail \
    -server smtp.domain.com \
    -l enfuzion@domain.com(Enfuzion Root) \
    -t "bob@domain.com(Bob),alice@domain.com(Alice)" \
    -s "Testing E-mail" \
    -body enfuzion.log
```

Enfnodescp

The EnFuzion **enfnodescp** program is the service control program for EnFuzion nodes on Windows. It provides service installation, uninstallation, start and stop.

The **enfstartup** program takes the following command line:

```
enfnodescp \
    install <service_exe>
    uninstall [<service_exe>]
    start [<service_exe>]
    stop [<service_exe>]
```

- **install <service_exe>**

  The command installs and starts the program in <service_exe> as a Windows service. The service is registered with Windows to execute at the boot time. The service is executed under the SYSTEM account. <service_exe> is normally the executable for the EnFuzion Starter Service.

- **uninstall [<service_exe>]**

  The EnFuzion node service is terminated and removed from the Windows service list. Any EnFuzion node processes are terminated as well. EnFuzion Starter Service is used by default, if <service_exe> is omitted.

- **start [<service_exe>]**

  The EnFuzion node service is started. The service must be installed, otherwise the command fails. EnFuzion Starter Service is used by default, if <service_exe> is omitted.

- **stop [<service_exe>]**

The EnFuzion node service is terminated, but it remains registered with the Windows system. Any EnFuzion node processes are terminated as well. EnFuzion Starter Service is used by default, if <service_exe> is omitted.

Enfnodeserver

The node server is the main process on the node. It receives jobs for processing from the EnFuzion root, controls their execution on the node and returns the results.

The node server can be started on a command line as:

```
enfnodeserver \
    [ -help ] \
    [ -a ] \
    [ -b ] \
    [ -c <port> ] \
    [ -d ] \
    [ -h ] \
    [ -id <string> ] \
    [ -j <number> ] \
    [ -n <host> <port> ] \
    [ -nb <host> <port> ] \
    [ -o ] \
    [ -p <hex_host> <port> ] \
    [ -r ] \
    [ -t <number> ] \
    [ -tl <seconds> ] \
    [ -v ] \
    [ -w <seconds> ] \
    [ -wd <directory> ]
    [ -wl <seconds> ] \
```

A description of options:

- **-help**

  Print out option descriptions.

- **-a**

  Execute in the autonomous mode. By default, the node server terminates all jobs and cleans the directories, if the connection with the root is terminated. In the autonomous mode, the node server keeps the jobs and the files and waits for another root connection (see the Section called *Bind* in Chapter 7).

- **-b**

  Execute in batch mode. By default, the node server exits after the connection with the root is terminated. In batch mode, the node server waits for another root connection (see the Section called *Batch* in Chapter 7).

- **-c <port>**

  <port> specifies the node server port number to which an EnFuzion root can connect (see the Section called *Node Port* in Chapter 7).

- **-d**

  Execute in daemon mode. This option is used when the node server is started from a remote system, so that the starting program can return immediately.

- **-h**

  Do not perform the initialization exchange with the starting program. This option is used primarily for internal EnFuzion purposes (see the Section called *Hello Message* in Chapter 7).

- **-id <string>**

  Specifies the node identifier. This is used during a restart of nodes that already have an identifier on the root. This option is used primarily for internal EnFuzion purposes.

- **-j <number>**

  Specifies the maximum number of concurrent executing jobs on this node (see the Section called *Requested Concurrent Jobs* in Chapter 7). The default value is 1.

- **-n <host> <port>**

  Specifies that the node server connects to the root and provides the root host name and the port number (see the Section called *Connect* in Chapter 7, the Section called *Connect Host* in Chapter 7, and the Section called *Connect Port* in Chapter 7).

- **-nb <host> <port>**

  Specifies a backup root host name and its port number, if the connection to the primary host is not successful (see the Section called *Connect Backup Host* in Chapter 7, and the Section called *Connect Backup Port* in Chapter 7).

- **-o**

  Prints out configuration and load monitoring options and exits. This option is useful for testing purposes.

- **-p <hex_host> <port>**

  Provides the host IP number and the port number for the job daemon on the EnFuzion root system. This option is used primarily for internal EnFuzion purposes.

- **-r**

  Do not report the node port number. This option is used primarily for internal EnFuzion purposes (see the Section called *Node Port Message* in Chapter 7).

- **-t <number>**

  Specifies the number of tries to connect to the EnFuzion root (see the Section called *Connect Retry* in Chapter 7).

- **-tl <seconds>**

  Specifies the time limit for the node server. After the node server execution exceeds the time limit, the node server stops requesting additional jobs and terminates after all the jobs on the node complete. (see the Section called *Execution Time Limit* in Chapter 7).

- **-v**

  Prints out a node server version and exits.

- **-w <seconds>**

  Specifies the delay between tries to connect to the EnFuzion root (see the Section called *Connect Delay* in Chapter 7).

- **-wd <directory>**

  Specifies the master directory for the node server. This option is useful for safely setting the node server working directory, for example when the node server is executed from a script or from a Java class.

- **-wl <seconds>**

  Specifies the time limit for the autonomous operation of the node server. The node server performs a cleanup and terminates all the jobs, if it is unable to connect to the EnFuzion root within this time (see the Section called *Wait Limit* in Chapter 7).

Enfpreparator

The Preparator allows you to build a plan without explicitly writing any EnFuzion commands. It is designed to allow easy creation of plans for the most common uses of EnFuzion.

The Preparator can be started on a command line as:

```
enfpreparator [ <plan_name> ]
```

The Preparator provides a wizard like interface which guides you through the process of plan creation.

More details about the Preparator are available in the Section called *The Preparator* in Chapter 8.

Enfprotectpass

The Enfprotectpass utility takes the file **enfuzion.nodes** in the current directory and produces a file with encrypted user accounts and passwords. The output file is named **enfuzion.nodes.e**. User accounts are replaced with "*" and passwords are replaced with a field, containing encrypted user accounts and passwords. The field starts with "***". Clear text passwords in the original configuration file can be changed to encrypted fields either by renaming the entire **enfuzion.nodes.e** file to **enfuzion.nodes** or by manually replacing clear text passwords with the corresponding encrypted fields. The default input and output file names can be changed through command line arguments.

The Enfprotectpass has the following command line arguments:

```
enfprotectpass \
    [ -v ] \
    [ -d ] \
    [ -i <file_name> ] \
    [ -o <file_name> ] \
    [ -s ]
```

- **-v**

  Print out the program version and argument descriptions.

- **-d**

  Read input from the standard input instead of the **enfuzion.nodes** file.

- **-i <file_name>**

  Read input from the file <file_name> instead of the **enfuzion.nodes** file.

- **-o <file_name>**

  Write output to the file <file_name> instead of to the **enfuzion.nodes.e** file.

- **-s**

  Write output to the standard output instead of to the **enfuzion.nodes.e** file.

More details about **enfprotectpass** are available in the Section called *Encrypted Passwords in enfuzion.nodes* in Chapter 6.

Enfpurge

The **enfpurge** utility takes a run file and its log and produces, on standard output, a run file consisting only of jobs that have not been completed. The output run file can be submitted to the Dispatcher to execute the remaining jobs.

The syntax of the Enfpurge utility is:

```
enfpurge <input_run> <log_file> <run_ID> > <output_run>
```

More details about **enfpurge** are available in the Section called *Enfpurge* in Chapter 9.

Enfreport

Enfreport has the following options:

```
enfreport \
    [ -type runs | nodes ] \
    [ -format text | csv | html ] \
    [ -root <working_directory> ] \
    [ -time <time_specification> ] \
    [ -columns <column_specification> ] \
    [ -group <name> ]
```

- **-type runs | nodes**

  This option selects the report type, which is either a run or a node report. The default value is **runs**. The report type determines values shown in the report. Run reports show node use by runs, and node reports show node utilization.

- **-format text | csv | html**

  This option selects the report output format, which is either text, HTML or CVS, comma separated values. The default value is text.

- **-root <working_directory>**

This option specifies the directory with the accounting information. The default option value is the **enfreport** working directory. Normally, the value of the root option would be the Dispatcher working directory, where accounting information is being stored automatically.

- **-time <time_specification>**

  This option selects the report time interval, which can be an hour, a day or a month. Hourly reports are available for the current and the previous calendar day, daily reports are available for the current and the previous calendar month and monthly accounts are kept indefinitely.

  <time_specification> is one of the following:

  - **-time H[[[<year>-]<month>-]<day>-]<hour>**

    produces an hourly report. If the year, the month or the day are omitted, the current date values are used.

  - **-time D[[<year>-]<month>-]<day>**

    produces a daily report. If the year or the month are omitted, the current date values are used.

  - **-time M[<year>-]<month>**

    produces a monthly report. If the year is omitted, the current date values are used.

  A report for the period between 12:00 and 13:00 of the current day is specified as "H12", whereas the same period on 30th of March 2001 should be specified as H2001-03-30-12. Similarly, a report for April 1st of this year would be denoted by time specification string "D04-01" and a report for the whole month of April would be specified as "M4".

- **-columns <column_specification>**

  This option selects the columns shown in the report. By default, all columns are shown.

  The <column_specification> string is a comma-delimited list of column definitions. Since spaces may be part of column names, make sure to include the string in quotes on the command line in order have it interpreted as a single command line argument. <column_specification> is one of the following:

  - **<column_name>**

    include the <column_name> in the report table;

  - **!<column_name>**

    exclude the <column_name> from the report table. If the "!" is the first item without <column_name>, then all columns are excluded.

- **<column_name>=<value>**

  include only rows where the value in the <column_name> matches <value>.

The list of available column names may be listed with the following commands:

```
enfreport -type runs -help columns
enfreport -type nodes -help columns
```

Enfreport prints the following column names that may be used in column definitions:

```
* Host Name
* ID
  Uptime
  Downtime
  Executing Time
  Idle Time
  Busy Time
  Jobs Done
  Jobs Started
  Avg Job Length
  Max Job Length
```

The columns marked with an asterisk "*" are key columns. If one or more key columns are excluded from the report, rows with same values of the remaining key columns are combined to one row.

- **-group <name>**

  selects only rows with users from this group.

More details on the EnFuzion service installation on Windows are provided in the Section called *The **enfreport** Program* in Chapter 9.

Enfstartup

The EnFuzion **enfstartup** program simplifies service installation on Windows. It provides service installation, uninstallation, start and stop.

By default, it uses the EnFuzion provided batch file, which is located in **config\enfboot.bat**. The Dispatcher is executed under the System account.

The **enfstartup** program takes the following command line:

```
enfstartup \
    install [<startup_script>]
    uninstall [<startup_script>]
    start [<startup_script>]
```

```
stop
```

- **install [<startup_script>]**

  The batch file in **<startup_script>** is registered with Windows to execute at the boot time. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory. Make sure that Windows is configured for starting programs at the boot time as described in the Section called *Network Service Installation* in Chapter 3.

- **uninstall [<startup_script>]**

  The batch file in **<startup_script>** is removed from files to execute at the Windows boot time. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory.

- **start [<startup_script>]**

  The batch file in **<startup_script>** is executed immediately. If **<startup_script>** is omitted, the default value is file **config\enfboot.bat** in the EnFuzion directory. If the Dispatcher is already running, then this command has no effect. To restart the Dispatcher, use **enfstartup stop** first, followed by **enfstartup start**.

- **stop**

  The EnFuzion root processes on the system are terminated.

More details on the EnFuzion service installation on Windows are provided in the Section called *Installing EnFuzion Root as a Network Service* in Chapter 3.

Enfsub

The **enfsub** program has the following options:

```
enfsub  [ <options> ]  <program>  [ <program_options> ]
enfsub  [ <options> ]  <script>  [ <script_options> ]
enfsub  [ <options> ]  [ -run ]  <run_file>  [ <input_files> ]
```

The program is used to submit the run for execution as a command line program, a script or a parametric execution, respectively.

<options> are:

- **-attach <run_ID>**

  attach to an existing run with the **<run_ID>** ID.

- **[-account | -a] <name>**

  a user specified string that is associated with the run for accounting purposes. The string can be used for generation of accounting reports.

- **-append**

  this is a switch for the **get** option. If the switch is present, then only new file content is retrieved and appended to the local file copy. Otherwise, the entire file is copied every time.

- **[-approval | -ap] [<n>][,<n>]...**

  approval jobs for the run. These jobs are scheduled first. After they complete, the run priority level is set to 10. The user needs to approve the run to return the priority level to its previous value. The run can be approved through the Eye. External tools can use the **run approve** API command to approve the run.

- **-completed**

  retrieves information about completed runs. This information is stored in the file **completed** in the **enfinfo** subdirectory of the current working directory.

- **[-count | -c] <number>**

  specify multiple jobs. This option can be used to execute the run multiple times. Jobs are distinguished by the environment variable ENFJOBNAME, which has a different value for each job. The option is used for command line programs or scripts. Run files already specify multiple jobs.

- **[-delete | -del]**

  delete a file from the EnFuzion root computer after it is fetched from the root computer to the local computer. By default, files are not deleted from the EnFuzion root computer. This option is used in conjunction with the **-fetch** option. If **-fetch** is not specified, then this option has no effect.

- **[-dir | -d] <path>[@<host_name>][,<path>[@<host_name>]]**

  specify the working job directory on nodes.

- **-e <user_name>@<host_name>,[<user_name>@<host_name>]**

  the list of recipients for e-mail notifications. Use the **-m** option to specify the condition for sending notifications.

- **[-export-environment | -x]**

  export the values of all environment variables from the submit host to the node.

- **-fail <number>**

  specify the maximum number of allowed failed jobs on a node. After <number> jobs fail on the node, no more jobs from the run are scheduled on the node.

- **[-fetch | -f]**

  fetch output files from the EnFuzion root computer. The output files are copied incrementally from the EnFuzion root computer to the submit computer as they are being created. This is useful for obtaining output files from completed jobs while there are still other jobs waiting or executing.

- **[-fetch-input | -fi]**

  fetch input files from the EnFuzion root computer. By default, only output files are being fetched. With this option, input files are being fetched as well. This option is used in conjunction with the **-fetch** option. If **-fetch** is not specified, then this option has no effect.

- **-get <file>**

  copy a file from the EnFuzion root computer to a local subdirectory. The file is copied to the **run-<runID>** subdirectory of the working directory.

- **-i <node_file>[=<submit_file>][,<node_file>[=<submit_file>]]**

  input files for the run. The files are first stored from the submit machine to the root machine and then made available to jobs on nodes.

- **[-localdir | -ldir] <directory>**

  change the default subdirectory for the **-rd** option. If **-rd** is not specified, then this command has no effect.

- **-login <file_name>**

  change the user identity to the one specified in the identity file **<file_name>**. The identity file is created with the **enfcmd identity** command.

- **-m [s | d | a | p | c]**

  the conditions to send e-mail notifications. **s** means execution start, **d** means execution done, **a** means execution abort, **p** means execution stop (pause), and **c** means execution approval (confirmation). Recipient addresses are specified with the **-e** option.

- **-max <number>**

  specify the maximum number of concurrently executing jobs for the run.

- **[-name | -n] <name>**

  the name of the run.

- **-nice [on|off][@<host_name>][,[on|off][@<host_name>]]**

  priority for execution of user jobs on nodes. A different option can be specified for different hosts. If nice is turned on, user jobs are executed at a background priority, allowing them to proceed only when the system would be otherwise idle.

  On Windows, nice executes processes at the IDLE_PRIORITY_CLASS class and THREAD_PRIORITY_ABOVE_NORMAL level. For example, a screen saver program on Windows is executed in same class but at a lower level THREAD_PRIORITY_NORMAL. On Linux/Unix, nice executes processes under the nice system call with the value of 10.

- **[-noautodetect | -nd]**

  disable automatic detection of input files. With this option, the parsing of the run file is disabled and only user specified files are copied.

  If this option is not specified and a run file is submitted, enfsub parses the tasks in the run file, identifies input files for the run and copies these input files from the submit computer to the EnFuzion root computer. These input files are copied in addition to any input files specified by the user on the command line. If an input file is specified in the run file, but does not exist, then the file copy is not attempted.

- **-o <root_file>[=<node_file>][,<root_file>[=<node_file>]]**

  output files from the run. The files are copied from nodes and stored in the result directory on the root.

- **[-poll-delay | -pd] <seconds>**

  the delay in seconds between contacting the EnFuzion root. The default value is 60s. For some operations, such as checking for run completion or new file, The **enfsub** program periodically contacts the EnFuzion root. This option changes the default interval between contacts.

- **[-quiet | -q]**

  disable the fetch progress report on individual files. By default, **enfsub** prints out files that are being copied from the EnFuzion root computer to the local computer under the **-fetch** option. This option disables these messages.

- **-rd**

  wait for the run to complete and copy run results to a separate run directory on the local host. This option can be used to include **enfsub** in scripts that submit a run and then process its results. By

default, the local directory is named **run-<runID>**. The default value can be changed with the **localdir** option.

- **-restart <number>**

  specify the number of times that a job can be rescheduled in the case of an error. When this number is reached, the job is terminated with an error.

- **[-results | -r]**

  wait for the run to complete and copy run results to the current working directory on the local host. This option can be used to include **enfsub** in scripts that submit a run and then process its results.

- **-root <host_name>:<port_number>**

  the address of the EnFuzion network service. The address can also be specified in the **submit.config** file. If the service address is not specified, a default value of **localhost:10102** is used.

- **[-start-time | -t] [[[[<year>-]<month>-]<day>-]<hour>:]<minutes>[.<seconds>]**

  specify the start time for the run execution. Run execution will be delayed until the start time.

- **-u <user_name>[@<host_name>][,<user_name>[@<host_name>]]**

  specify user accounts for job execution on nodes.

- **[-value | -v] <name>[=<value>][,<name>[=<value>]]**

  specify environment variables and their values.

- **[-wait | -w]**

  wait for the run to complete. The **enfsub** program will not return until the run is completed. This option can be used to include **enfsub** in scripts that submit a run and then process its results.

- **[-wall-time | -wt] <hour>[:<minutes>[:<seconds>]]**

  the maximum wall time interval that the run is allowed to execute.

More details about **enfsub** are available in the Section called *The Enfsub Program* in Chapter 10.

Netsetup

The Netsetup program can be used in Windows environments to install EnFuzion on remote systems, without any need to access the system's keyboard or monitor. The program can also be used to control the EnFuzion Starter Service on remote computers.

The Netsetup program is called with a set of options, followed by a command and command options:

```
netsetup [ <option> ] <command> <options>
```

The following are **netsetup** options:

- **-v**

  Prints the **netsetup** program version and options.

- **-d**

  Reads EnFuzion nodes from standard input instead of from the file **install.nodes**.

- **-p**

  Prints command progress.

- **-t <number>**

  Executes the command concurrently on at most <number> hosts. The default value is 1, so the command is executed sequentially for each host.

The following are **netsetup** commands:

- **install \\<host>\<share>\<source> <destination>**

  Installs EnFuzion executables from a source directory to the destination directory on hosts specified in the file **enfuzion.nodes**. Options are as follows:

  - **<host>** is the name of the host where the EnFuzion package has been unpacked and has been made available for access over the network.
  - **<share>** is the name of the share on the <host>, which contains the <source> directory.
  - **<source>** is the directory containing the setup program and other EnFuzion distribution files.
  - **<destination>** is required for the initial EnFuzion installation. It specifies the EnFuzion installation directory. Its recommended value is **C:\enfuzion**. <destination> is not required, if EnFuzion is already installed on systems.

- **uninstall**

  uninstalls EnFuzion from all hosts.

- **start**

  starts the EnFuzion Starter Service.

- **stop**

  stops the EnFuzion Starter Service.

- **delete**

  deletes the EnFuzion Starter Service from the service control manager database.

- **verify**

  prints EnFuzion Starter Service status information.

More details about **netsetup** are available in the Section called *The Netsetup Program* in Chapter 3.

Setup

The EnFuzion installation and upgrade program for Windows is called **setup**. Most often, the user executes the program by clicking on the file. In that case, **setup** asks for any user options and installs EnFuzion software on the system. The program also provides additional command line options, which are useful for remote and automated management. This section provides details on the **setup** options.

The **setup** program takes the following command line:

```
setup [ <options> ]
```

- **-main <directory>**

  Define the main EnFuzion directory. Default value is **C:\enfuzion**. If EnFuzion is already installed on the system, this option has no effect.

- **-tmp <directory>**

  Define the EnFuzion temporary directory. Default value is **C:\enfuzion\temp**. If EnFuzion is already installed on the system, this option has no effect.

- **-node**

  Install only EnFuzion node components.

- **-root**

  Install only EnFuzion root components.

- **-submit**

  Install only EnFuzion submit components.

- **-force**

  Force program installation. By default, **setup** does not overwrite an executable file, if it is being used by a process. With this option, the program is terminated, so that the installation of the file can be completed successfully.

- **-noprompt**

  Use default values. With this option, **setup** does not request any input from the user. The program uses default values to perform the installation.

- **-s**

  Perform a silent installation. Do not produce any output and do not request any input from the user.

- **-ignore**

  Ignore errors during program installation. This option is applicable for EnFuzion upgrades. If an EnFuzion program is executing during an upgrade and the **-s** option is turned on, the **setup** program terminates by default. With this option, any errors while upgrading executing programs are ignored and the upgrade proceeds.

More details about the installation of EnFuzion on Windows is provided in Chapter 3.

Starter Service

The EnFuzion Starter Service runs on each EnFuzion node as a service. It provides remote access to a Windows NT/2000/XP host. Its primary function is to start remote execution.

The Starter Service uses the IP port number 17000 to listen for user requests.

The Starter Service provides remote management commands. These commands are ASCII strings, terminated by a null character, '\0'. Supported commands are:

- **version**

  Returns the current Starter Service version, terminated by a newline character, '\n', followed by a null character, '\0'.

  >Example of a return string:

  ```
  7.2.30\n\0
  ```

- **clearlog**

  Truncates the Starter Service log file in **enfstarter.log**. It returns the string "OK\n\0" if the log was truncated. Otherwise, it returns:

  ```
  Unable to clear log file "....\enfstarter.log".\n\0
  ```

  Example of a return string:

  ```
  OK\n\0
  ```

- **getlogs**

  Returns the contents of two node log files. The **enfnodea.log** is printed first, followed by the **enfnodeb.log** file. If the log files do not exist, it returns:

  ```
  Unable to copy file enfnodea.log\n\0
  ```

  See the Section called *Log File Size* in Chapter 7 for more details about the node log files.

More details about the Starter Service are available in the Section called *Starter Service* in Chapter 3.

Uninstall

The program Uninstall, which is located in the EnFuzion directory, removes the EnFuzion Starter Service from the system, deletes EnFuzion files, directories and registry entries. Any user files in the EnFuzion directory are not affected. Uninstall is supported only on Windows platforms.

More details about the **uninstall** are available in the Section called *Removal of EnFuzion Software from Windows NT/2000/XP* in Chapter 3.

Appendix A. Frequently Asked Questions

1. EnFuzion root programs are not working. How can I proceed?

One common installation error is having an incorrect execution path. You need to make sure that all EnFuzion programs are on the execution path on the root computer.

Verify that all the EnFuzion executable files are in a directory which is on your execution path.

On Unix, installation files are initially located in the package directory named **enfuzion.<version>-<os>.<osversion>-<processor>**.

<osversion> corresponds to the operating system. <processor> corresponds to the processor type of your root machine.

On Windows NT/2000/XP, the default installation directory for EnFuzion root, used by **setup.exe**, is: **C:\enfuzion**.

2. An EnFuzion node is not working. How can I proceed?

On Unix, EnFuzion will search for the node executables in the directory **~/enfuzion** for ordinary users and in **/usr/local/enfuzion** for the root user.

On Windows NT/2000/XP, EnFuzion will search for the node executables in the directory **bin** in the main **EnFuzion** directory. If the executables are not found in this directory, then they must be accessible through the execution path.

On Unix, you can test that node executables are accessible by running the command **enfinstall verify**. If this command does not work, then login into each node using **telnet** and test the path by typing:

```
<dir>/enfnodeserver -v
```

Replace the <*dir*> with a valid directory for your configuration. This command will report the current version of the node.

If the node is not started, then make sure that its executable is in the expected directory and that its execution permissions are set.

3. The license is not working. How can I proceed?

On Unix, EnFuzion will search for the license file **enflicense** in the following directories: current directory, directories in the execution path, **~/enfuzion**, and **/usr/local/enfuzion**.

On Windows NT/2000/XP, EnFuzion will search for the license file **enflicense** in the following directories: current directory, main EnFuzion directory.

Make sure that a valid license is placed in one of the valid directories. One common error is to have an obsolete license in a directory that is in before the more current license in the search order above.

4. Load monitoring is not working. How can I proceed?

Make sure that the **enfuzion.options** file is installed. On Unix, EnFuzion will search for the system **enfuzion.options** file in the directory **/var/opt/enfuzion**, and for the user file in directory **~/enfuzion**. On Windows NT/2000/XP, EnFuzion will search for the **enfuzion.options** file in the main EnFuzion directory.

The node has an option that allows you to verify load monitoring options. Login to the node using **telnet** and display the options by typing:

```
<dir>/enfnodeserver -o
```

Replace the <*dir*> with a valid directory for your configuration. This command prints out load monitoring options as seen by the node.

5. My application is not executing properly on nodes. What should I do?

EnFuzion provides extensive reporting of system and user errors.

Most common execution errors are reported in the log file, called **enfuzion.log**. The log file contains error reports and diagnostic messages.

Execution errors by user applications are reported to their standard output and standard error. These files are automatically copied from a node to the root computer, if one of the commands in the plan fails. The files can be of great assistance in determining the nature of the errors. See the Section called *Handling of Network Failures* in Chapter 1.

Make sure that the application is capable of running on all the node computers as specified in the configuration file. The application must be accessible through the execution path on the node, or copied to the node as part of the job execution.

If the application is to be accessible through the execution path, you can login to a node which causes problems, and try running the application from the command line. If this is not working, modify your execution path or install the application on the node.

One common error is to forget some input files that are required by the application. Make sure that all input files are either copied to nodes as part of the job execution, which is specified in the plan, or are accessible on the node locally or via NFS.

Try to avoid referring to files using relative pathnames through parent directories. On node computers, user jobs execute in their own directories, and the location of these directories might change in the future. It is recommended that file names are specified as files or subdirectories in the local directory or as absolute path names. The EnFuzion **copy** command will copy files relative to the local directory.

6. Does EnFuzion require Windows NT Server for its operation?

No. EnFuzion works with Windows NT Workstation and Windows NT Server.

See also Chapter 3.

7. Does EnFuzion work in mixed Unix and Windows NT/2000/XP networks?

Yes. EnFuzion works in heterogeneous networks. It can combine a large number of Unix and Windows NT/2000/XP computers to work as a single cluster.

See also the Section called *Installation in a Mixed Linux/Unix and Windows NT/2000/XP Environment* in Chapter 4 and the Section called *Installation in a Mixed Windows NT/2000/XP and Linux/Unix Environment* in Chapter 3.

8. How can I configure EnFuzion to use Linux/Unix and Windows NT/2000/XP at the same time?

See the Section called *7. Does EnFuzion work in mixed Unix and Windows NT/2000/XP networks?* above.

If an EnFuzion root on Unix is using an EnFuzion node on Windows NT/2000/XP, this needs to be specified in the network configuration file **enfuzion.nodes** with an additional "WindowsNT" keyword for each Windows NT/2000/XP host. A line in **enfuzion.nodes** would thus look as follows:

host user password WindowsNT

Similarly, an EnFuzion root on Windows NT/2000/XP requires keyword "Unix" for each Unix node:

host user password Unix

These additional keywords are necessary because remote execution on Unix and Windows NT/2000/XP is implemented differently. If the root and the node are on hosts of the same type, then these keywords are not required.

9. I am unable to access a Windows NT/2000/XP network drive.

Or, I am unable to access a Windows NT/2000/XP network drive from my application that is executing on an EnFuzion node. How can I overcome this problem?

By default, Windows NT/2000/XP will not map network drives if nobody is logged on the machine. Since EnFuzion node programs appear to Windows NT/2000/XP as batch processes, some network drives might not be seen by EnFuzion jobs.

The simplest solution is to map the network drives with an additional command in EnFuzion plans using the **execute** command. The Windows NT/2000/XP command **net use** explicitly maps network drives for local access. For example, the following EnFuzion command, using the **net use** command will map a network drive to local "z:" drive:

```
node:execute net use z: \\computername\sharename
```

10. Can I avoid plain text passwords in the network configuration file enfuzion.nodes?

Yes. It is possible to encrypt passwords using the Enfprotectpass utility. The utility Enfprotectpass takes the file **enfuzion.nodes** from its working directory and produces a file with encrypted passwords and other user information. The output file can be renamed to **enfuzion.nodes** and used instead of the original file. By default, the output file is named **enfuzion.nodes.e**. A user can change the name of the output file with the **-o** option.

See the Section called *Encrypted Passwords in enfuzion.nodes* in Chapter 6.

11. How can I configure EnFuzion to avoid conflict with a user working on a node?

I will be using EnFuzion to distribute calculations during night or weekends to fully use the idle CPU time. How can I set up EnFuzion so that a user who is using a computer interactively will not be affected by my calculations?

EnFuzion has extensive support for load monitoring on local hosts. See the Section called *Specifying Load Monitoring Options* in Chapter 7. It is able to detect interactive users and execute jobs only on hosts which are idle.

For more details, see the Section called *Screen Saver* in Chapter 7.

12. How can I configure EnFuzion to execute two simultaneous jobs on a dual processor host?

The maximum number of concurrent jobs to be executed on the specified node can be specified by the **joblimit** option.

See Chapter 7.

13. How do I manually install EnFuzion on Linux/Unix?

Refer to Chapter 4.

14. What are the default installation directories under Unix?

See the Section called *Directory Layout* in Chapter 1 and the Section called *Directory Layout* in Chapter 1.

15. The installation program on Linux/Unix complains about incorrect user or password on a remote machine. What should I do?

Verify that you can manually connect via **telnet** and **ftp** to the remote machine. The installation program on Linux/Unix requires that **telnet** and **ftp** servers are both executing on the remote machine. Sometimes, a machine will allow connections from one but not the other. Make sure that you test both.

If **ftp** is not allowed, then you can install EnFuzion manually. EnFuzion itself will function even without an **ftp** connection.

If **telnet** is not allowed, more secure protocols (such as **ssh**) can be used with EnFuzion to access the nodes.

16. How does EnFuzion on Linux/Unix communicate with remote machines?

The Linux/Unix installation program, Enfinstall, by default uses standard Unix commands **telnet** and **ftp** to install EnFuzion on remote machines. No special software is required on the EnFuzion root machine, because **telnet** and **ftp** clients are implemented as part of EnFuzion. EnFuzion node machines must have **ftp** and **telnet** servers running. They must be accessible via **ftp** and **telnet** commands from the EnFuzion root machine.

Apart from **telnet**, EnFuzion can use other protocols such as **ssh** or **rsh** to access node machines.

The Dispatcher automatically starts EnFuzion nodes via **telnet** when required. After the connection between the EnFuzion root and an EnFuzion node is established, EnFuzion uses the TCP/IP protocol to transfer messages. If it is necessary to copy files, these will be copied by EnFuzion over TCP/IP. Alternatively, NFS can be used by specifying appropriate path names in EnFuzion run files.

See the Section called *Root - Node Communication* in Chapter 1.

17. How does EnFuzion compare to batch queue managers?

Batch queue managers are specialized to share the computational load by executing individual tasks on the most appropriate computer. Batch queue managers do not provide facilities for the generation of jobs nor facilities for the management of a large number of jobs belonging to a single application.

EnFuzion is complementary to batch queue managers. It can be used stand-alone or in conjunction with batch queue managers. EnFuzion divides a large task into a number of smaller jobs and then uses all available computing resources to execute the jobs as fast as possible. It will thus keep all the available computers fully utilized. EnFuzion is application oriented, since the jobs will be managed in the context of a single application.

EnFuzion supports parametric execution. With parametric execution, input parameters are varied, but the program to be executed remains the same. Each set of input values generates one job. Variations in input parameters usually produce a large number of jobs. This large number of jobs and resulting outputs, sometimes exceeding thousands, is hard to manage and takes a long time to compute.

EnFuzion provides significant benefits for parametric execution:

- It radically simplifies the generation and distribution of jobs and the collection of job results.
- Because it distributes jobs over a network of computers in a user transparent fashion, the jobs are computed much faster than on a single computer.

EnFuzion thus greatly simplifies and speeds up parametric executions. Batch queue managers address only the distribution aspects of parametric executions, but do not provide any help with the job generation or management aspects.

Although EnFuzion provides its own job distribution mechanism, it can be integrated with other batch queue managers, if that is required. In that case, EnFuzion submits jobs through a batch queue manager. This allows smooth integration of EnFuzion with existing load distribution policies. Axceleon will be happy to provide additional information on the integration of EnFuzion with batch queue managers. Send e-mail to info@axceleon.com.

18. Where can I learn about the early technology behind EnFuzion?

For more information on some of the technology behind EnFuzion and its related research project Nimrod, see Nimrod. (http://www.csse.monash.edu.au/~davida/nimrod.html)

Index

communication port, 111
access control
  Eye , 250
API, 11, 85, 186, 196, 199, 268
  from C, 282
authentication
  primitives, 144
backupport option , 113
batch option, 114
bind option, 90, 115
cluster, 1, 98, 189
  communication channel, 269
  defined, 6
  directory, 192
  information, 257
  log records, 212
  monitoring status , 227
  nodes, 4
  object type, 268
  options, 190
  parameters, 194
  with large number of nodes, 77
cluster commands, 270
cluster event reception, 270
cluster object
  variables, 191
Cluster Status page
  Eye , 228
clusters, 5, 79
command line interface, 5
command line program, 1
conditional statements, 183
configuration files, 11, 14, 15, 17
configuration option, 3
connect backup host, 112
connect delay option, 114
connect host option, 112
connect option , 111
connection commands, 281
connection object type, 268
connectretry option, 113
context, 3, 190, 193, 196
  options, 193
context object
  type, 268
datajobs , 3, 99, 180, 195, 199, 200
  executing, 200
  format, 200
  log records, 212
  output, 200
  overview, 179
  port connection, 179
  request times, 177
  static, 199
  streaming, 199
  timeout, 198
decryption, 17, 140, 140, 147
  user defined, 138
decryption primitives, 138
Detailed Node Information page
  Eye, 236
Detailed Run Information page
  Eye, 230
direct command, 269
directories
  custom run, 207, 227
Dispatcher, 11, 15, 56, 211
  connecting from C, 282
  daemon mode, 202, 289
  defined, 9
  deleting completed directories, 198
  enfuzion.nodes file, 74
  linking with job server, 200
  log, 10, 212
  messages to Director, 269
  multiple run mode, 9, 205
  node subdirectory, 16
  options , 202, 289
  overview , 201
  persistent runs, 195
  port connection, 91
  provides the API, 268
  provides the HTTP API, 260
  required task descriptors, **??**
  server command, 179
  single run mode, 9, 203, 290
  transient runs, 196
  working directory, 14, 190
dynamic library, 15, 138, 144
enfacct
  command reference , 287
enfcmd
  command reference , 288

monitoring run results with, 215
enfdispatcher
program reference , 289
enfexecute
program reference , 292
enfgenerator
program reference, 293
enfinstall
program reference , 294
enfkill
program reference , 295
enfmail
program reference , 295
enfnodeserver
program reference, 298
enfpreparator
program reference, 301
enfprotectpass, 107, 301
program reference, 301
enfpurge
program reference , 302
Enfsub
overview, 207
enfuzion.log, 14
enfuzion.nodes file
overview, 5
errors
detection, 12
Eye messages, 247
job execution, 19
system, 19
user, 19
executables, 14
installed by Enfinstall, 66
node, 16, 17
password encryption, 136
root, 15
trusted, 136
execution
environment, 186
search path, 18
Eye
access control, 250
accessing via proxy, 207, 227
browsers supported, 223
Cluster page , 214
Cluster Status page, 228
connecting to EnFuzion, 223
default port number, 207
Detailed Node Information page, 236
Detailed Run Information page, 230
error messages, 247
home page , 224
monitoring cluster status, 227
monitoring run results, 240
Node List page , 214, 235
overview, 223
program reference, 292
Run List page , 215, 229
Single Node page , 214
Single Run page , 215
starting, 223
submitting runs, 207, 226
Generator
batch mode, 165
defined, 165, 293
interactive mode, 165
starting, 165, 294
hardware requirements, 3, 5, 6
heartbeat, 4, 11, 12, 97
hello option, 116
Input Files, 159
input values
specifying, 165
interface
web based, 223, 292
job, 1, 2, 186, 188, 190, 196
commands , 279
daemon, 194
daemon port, 191, **??**
directory , 17, 18
events logging, 212
execution
abort, 280
concurrent, 198
directory, 18
errors, 197
timeout, 197
multiple executions of, 13
object type, 268
options, 193
output file, 18
output files, 14
overview, 3
parameter as a string, 184
parameters, 186, 189, 194

priority, **??**
ready queue, 280
requirements, 196
server , 18, 200
stop action, 128
subdirectory, 17
tasks, 168, 171
termination signal, 130
throughput rate, 1
variable definition example, 181
variables, 187
job submission, 4
jobs, 192, 196, 212
automatic resubmit, 12
batch queue submission, 320
common files, 16, 18
concurrent maximum number of, 318
datastream, 3
default priority , 121
default task main, 172
enfpurge , 211, 302
error detection, 12
examples, 188
execution priority, 195
execution limit, 197
execution overview, 8, 9
global variables, 180
idle time, 122
input values, 172
lightweight, 179
log of successfully completed, 211
off periods, 95
partaking of run scope, 180
regular, 3, 199
requested concurrent , 110, 129
requested maximum number of, 118
requirements, 196
screen saver active, 121
streams, 196
two types of, 3
unique identifier, 19
variable changing, 180
variables scope local, 180
keywords, 317
data, 200
Unix, 317
WindowsNT, 317
library template, 141, 147
load monitoring, 6, 67, 117, 178
options, 6
locators, 173
logs, 98, 146, 212
cluster logs, 269
enfuzion.log , 212
run logs, 212
main task , 119
multiple execution, 12, 198
netsetup
program reference, 310
network based API, 4, 5
network service, 4
Nimrod, 320
node
commands , 273
object type, 268
options, 191
parameters, 194
port message option, 116
port option, 111
properties, 191, 196
startup, 171
node host, 2, 22, 27
Node List page
Eye , 235
node server
overview , 6
starting, 298
node.config file, 110
overview, 6
sample, 116
nodes, 18, 51, 68, 186
allocated number of, 195
concurrent activation, 96
concurrent execution on, 198
directory layout, 16
enfinstall, 66, 67
enfuzion.nodes , 106, 203, 290
failed nodes, 12
overload prevention, 118
overview , 6
properties, 196
starting, 11, 12
Windows nodes, 176
nodestart task, 16, 119
observe command, 269
off on periods, 126, 191

options, 118
options file
  node.config , 110
Output Files, 159
parameter , 3
parameters, 1, 3, 147, 187, 189, 190, 268
  list of, 187
parametric execution, 1
  overview, 2
persistence, 195
plan
  example, 160
plan files
  comments, 168
  configuration options, 168
  converting to run files, 9
  defined, 153
  described, 167
  overview , 9
  parameter statements, 168
  parameters, 167, 168
port
  node options, 111
post processing command, 159
predefined tasks, 171
preemption, 195
Preparator
  creating plan files, 153
  creating plan files with, 156
  defined, 156, 301
  starting, **??**, 301
  wizard, 157, 301
preprocessing command, 159
remote execution, 11, 79, 312
resource management, 1
result retrieval, 4
root, 4, 68, 127, 142
  authentication , 136, 142, 144, 146, 147
  configuration overview, 5, 8
  enfdispatcher, 5
  environment, 14
  executables, 15
  execute user commands on, 176
  file references, **??**
  installing on Linux/Unix, 57
  installing on Windows, 41
  job daemon executes on, 191
  job server executes commands on, 18
  locators, 173
  permanent connection with node, 11
  root.options, 85, 190
  specifying root hosts, 137
  startup, 171
root host, 2, 22, 27
rootport option , 112
run, 1, 2, 13
  adding a job, 276
  adding a task, 277
  approval
    abort, 275
  commands , 274
  completed directories, 198
  context properties, 280, 281
  datajobs, 199
  defined, 3
  definition, 275
  directory, 14, 19, 191
  empty, 271
  enfuzion.options copying, 118
  execution
    abort, 275
    limit , 197
    start, 274
    stop, 275
  file, 179, 186
    Enfpurge created , 211
    enfpurge example , 211
  files, 183, 319
  id, 194, 257, 259, 271
  identifiers using enfcmd, 258
  jobs input values (parameters), 172
  level, 195
  level requirements, 196
  licenses, 192
  log, 14, 98, 211, 212, 212
  name, 194
  object type, 268
  options, 192
  overview, 2
  parameters, 194
  persistence, 191, 195
  preemption, 191, 195
  priority, 186, 194
  removing, 272
  requirements, 192, 197
  reschedule

abort, 275
scope, 180
scope of variables, 189
subdirectory, 16, 192
transience, 191
variable , 19, 179, 190, 194
weight, 195
working directory , 272
run execution limit, 191
run files
defined, 153
name, 271
overview , 9
phases for creating, 156
Run List page
Eye , 229
run results
files , 217
monitoring from a web browser, 214
monitoring from command line, 215
monitoring from custom program, 216
run variable example, 198
runs
description overview , 9
execution overview , 9
monitoring execution, 10
monitoring results of , 240
retrieving results , 10
submission steps , 207
submitting from custom program, 210
submitting with the Eye , 207, 226
scope , 180, 189, 190, 190, 193
script, 1
scripting languages, 4, 185
set command, 180
software requirements, 3, 5, 6
ssh, 319
Starter Service, 12, 42, 49, 51, 53, 139, 310, 312
program reference, 312
submission control, 4
submission monitoring, 4
submit host, 2, 22, 27
submit hosts, 3
task, 3, 18, 69, 172, 179, 185
common, 3
description, 3
enfexecute task_command, 185, 292
format , 171
main, 17, 119, 172
name, 187
nodestart, 16, 118, 172
onerror, 172
rootfinish, 172
rootstart, 171
server, 200
task file example, 172
tasks, 68, 119, 139, 144, 168, 171, 190, 259
predefined, 171
TCP/IP protocol, 1, 11, 12, 41, 268
telnet, 12
throughput, 1
timelimit option, 114
timeout, 180, 189, 198, 198
transient run, 196
uninstall
program reference , 313
user, 2, 7
privileges, 7
User Commands, 159
user ID, 2, 7
anonymous, 7
user passwords , 317
variables, 186, 190, 268
as options, 189
as parameters, 189
environment, 186
node, 190
root, 190
waitlimit option, 90, 115
web based interface, 5
wizard
dialogs, 157
Finishing dialog, 159
Input Files dialog, 159
Output Files dialog, 159
Parameter Description dialog, 158
Post processing dialog, 159
Preparator , 301
Preprocessing dialog, 159
User Commands dialog, 159